

ALGORITMOS E TÉCNICAS DE PROGRAMAÇÃO

Unidade 1

Fundamentos dos Algoritmos e das Linguagens de Programação

Aula 1

Introdução aos Algoritmos

Introdução aos algoritmos



Este conteúdo é um vídeo!

Para assistir este conteúdo é necessário que você acesse o AVA pelo computador ou pelo aplicativo. Você pode baixar os vídeos direto no aplicativo para assistir mesmo sem conexão à internet.

Dica para você

Aproveite o acesso para baixar os slides do vídeo, isso pode deixar sua aprendizagem ainda mais completa.

Olá, estudante! Nesta videoaula, você verá os conceitos essenciais dos algoritmos, vendo os diferentes tipos e revelando aplicações práticas que permeiam nosso cotidiano.

Além disso, abordaremos assuntos relativos à introdução aos algoritmos como: conceitos, tipos de algoritmos e a aplicação deles em exemplos do mundo real, fundamental para sua compreensão.

Não perca a oportunidade de aprimorar seus conhecimentos e expandir sua compreensão sobre algoritmos. Vamos lá?

Ponto de Partida

Olá, estudante.

A partir deste momento, exploraremos juntos os princípios dos algoritmos, a base sólida de qualquer programador. Ao embarcar nesta jornada, desvendaremos os segredos por trás da construção de soluções eficientes para problemas computacionais.

ALGORITMOS E TÉCNICAS DE PROGRAMAÇÃO

No decorrer da disciplina serão abordados conteúdos que irão introduzir você ao mundo da programação. Vamos explorar os diversos tipos de algoritmos, entendendo suas características únicas e como cada um deles se encaixa perfeitamente em diferentes contextos.

Além disso, ao final desta Aula, você vai compreender teoricamente o que são algoritmos, e ainda, será capaz de visualizar como aplicá-los em situações do mundo real.

Atualmente o desenvolvimento de sistemas e aplicativos ligados a programação de computadores está presente na sociedade. O desenvolvimento do raciocínio lógico é fundamental para que o profissional se desenvolva em qualquer área do conhecimento que deseja atuar.

Diante disso, será apresentado o desenvolvimento de um algoritmo com uma tarefa bem comum: a preparação de uma xícara de café por meio de passos simples que tornarão possível esta atividade.

Convido você a iniciar a construção de uma base sólida para sua jornada no universo da programação. Preparado para essa imersão?

Vamos Começar!

O objetivo desta aula é apresentar a introdução à algoritmos, seus conceitos e definições, suas principais características, além de iniciar você estudante neste enorme cenário de tecnologia em que estamos inseridos.

Aqui, exploraremos as bases que sustentam a magia por trás dos programas de computador, os algoritmos. Antes de abordarmos os conceitos de lógica de programação, é importante entender o que nos aguarda nesta viagem.

Em um vasto campo, onde a lógica é a bússola e os algoritmos são as trilhas que podemos percorrer, esses algoritmos, como mapas, guiarão nosso raciocínio na resolução de problemas, construção de soluções e criação de programas eficientes.

Conceitos e introdução aos algoritmos

Iniciando este tópico, vamos começar apresentando um pouco dos conceitos que envolvem os algoritmos. Mas o que são Algoritmos?

Os algoritmos são sequências lógicas de passos ou instruções, representando um conjunto de operações bem definidas e ordenadas para resolver um problema específico. Esses procedimentos, essenciais no campo da ciência da computação, formam a base para o desenvolvimento de programas e soluções computacionais.

ALGORITMOS E TÉCNICAS DE PROGRAMAÇÃO

Em essência, um algoritmo é como uma receita que guia a execução de tarefas. Assim como um cozinheiro segue passos precisos para preparar um prato, um programador cria algoritmos para alcançar resultados desejados em um contexto computacional. Cada passo deve ser claro, preciso e capaz de ser executado de forma determinística.

A versatilidade dos algoritmos permite que eles se adaptem a uma variedade de domínios, desde tarefas simples até problemas complexos. Seja ordenando números, buscando informações em grandes conjuntos de dados ou controlando um robô, os algoritmos são a linguagem universal da computação, traduzindo lógica e raciocínio em ações concretas.

Ao compreender os algoritmos, desvendamos os segredos por trás da eficiência computacional, da resolução de problemas e do desenvolvimento de software. Em última análise, os algoritmos capacitam a transformação de ideias em ações, revelando-se como o cerne da inteligência computacional.

A Lógica, do ponto de vista filosófico é um conjunto de ações que determinam processos intelectuais que são condições gerais do conhecimento verdadeiro. Essa definição vem da lógica aristotélica (FERREIRA, 1999). Em outras palavras, podemos definir a lógica como um conjunto de tarefas organizadas e estruturadas com um propósito bem definido para solução de um problema.

No seu livro *Estudo Dirigido de Algoritmos*, Manzano; Oliveira (2012, p. 14) define a lógica como:

uma sequência coerente, regular e necessária de acontecimentos, de coisas e na área da informática é a forma pela qual assertivas, pressupostos e instruções são organizados em um algoritmo para implementação de um programa de computador.

Diante destes conceitos de lógica apresentados, partimos então para os algoritmos que nada mais são do que as trilhas sinalizadas através da bússola, ou seja, a nossa lógica.

Tipos de algoritmos

Diante destes conceitos de lógica apresentados, partimos para os algoritmos que são as trilhas sinalizadas através da bússola, ou seja, a nossa lógica.

Explicando melhor, um algoritmo é a solução de um problema ou são os passos necessários para a conclusão de uma tarefa. Por exemplo: Digamos que você tenha que trocar o pneu de um carro. Logicamente existe uma sequência de instruções que deverão ser seguidas para a conclusão dessa tarefa. Isso é um Algoritmo. Dessa maneira observe as instruções desse algoritmo no Quadro 1, a seguir:

ALGORITMOS E TÉCNICAS DE PROGRAMAÇÃO

	Estacione veículo em local seguro, desligue o motor e acione os sinais de emergência
2	Ferramentas Prontas
	Certifique-se de ter as ferramentas necessárias: chave de roda, macaco e estepe
3	Elevação do Veículo
	Use o macaco para levantar o veículo, seguindo as instruções do manual do carro
4	Troca Rápida
	Remova a roda danificada e substitua pelo estepe. Aperte os parafusos
5	Finalização
	Baixe o veículo com cuidado, aperte os parafusos novamente e verifique a pressão do pneu substituto

Observe no Quadro 1 que as instruções de 1 a 5 são claras quanto a realização da atividade da troca de pneu. Seguindo esses passos é possível atingir o objetivo que é o resultado do algoritmo.

Depois que o algoritmo está pronto, apenas precisamos segui-lo. Estamos certos se pensamos assim, porém devemos lembrar que para desenvolver esse algoritmo, esses passos para realizar essa ou qualquer outra tarefa, estamos exercitando nosso raciocínio lógico e nossa capacidade intelectual.

Por esse motivo, o desenvolvimento do raciocínio lógico é um dos objetivos e disciplinas ligadas a lógica de programação e aos algoritmos.

Ao adentrarmos no universo da programação, deparamo-nos com desafios que exigem soluções bem estruturadas. Para tal, é importante compreender as distintas formas de representação de algoritmos. Ao todo são 3 (três) formas, a saber:

- Narrativa.
- Fluxograma.

ALGORITMOS E TÉCNICAS DE PROGRAMAÇÃO

- Pseudolinguagem.

Vamos abordar cada um desses três pilares de forma mais detalhada.

Forma Narrativa

Em nossa jornada, deparamo-nos, inicialmente, com o algoritmo narrativo, uma narrativa textual que guia o profissional que vai desenvolver o algoritmo pelos passos necessários para alcançar a solução.

Nesse ponto, utilizamos uma linguagem próxima ao nosso cotidiano, descrevendo de maneira clara e sequencial as ações a serem executadas. Essa forma de representação, embora acessível, torna-se mais desafiadora à medida que lidamos com problemas complexos, em que a clareza verbal pode encontrar limitações. Isso acontece pelo problema de comunicação, pois as vezes a interpretação não é realizada corretamente.

O exemplo da troca de pneu exibido no Quadro 1 é definitivamente um algoritmo representado pela forma narrativa. Acompanhe este algoritmo com algumas variações sendo um exemplo deste primeiro pilar de representação de um algoritmo.

Preparação:

- Estacione o veículo em local seguro e plano.
- Desligue o motor e acione o freio de estacionamento.
- Pegue o macaco, chave de roda e o estepe do porta-malas.

Elevação do Veículo:

- Localize o ponto de elevação no chassi do veículo.
- Posicione o macaco sob o ponto de elevação.
- Gire a manivela do macaco para elevá-lo até que a roda saia do chão.

Remoção da Roda Danificada:

- Utilize a chave de roda para afrouxar as porcas da roda.
- Remova a roda danificada.

Instalação do Estepe:

- Coloque o estepe no lugar da roda danificada.
- Rosqueie as porcas manualmente até que fiquem ajustadas.

Baixando o Veículo:

ALGORITMOS E TÉCNICAS DE PROGRAMAÇÃO

- Gire a manivela do macaco no sentido oposto para baixar o veículo.
- Aperte completamente as porcas com a chave de roda.
- Guarde as ferramentas no porta-malas.

Fluxograma

O fluxograma é uma ferramenta gráfica poderosa que traduz a lógica de programação em um mapa visual compreensível. Cada elemento gráfico em um fluxograma representa uma instrução específica, decisão ou processo, criando uma representação visual clara do algoritmo. Essa abordagem torna-se especialmente útil ao lidar com algoritmos complexos, permitindo uma análise detalhada de cada etapa (MANZANO; OLIVEIRA, 2012).

No fluxograma, diferentes formas são usadas para denotar diferentes elementos. O início e o fim do algoritmo são representados por elipses, enquanto retângulos são usados para indicar instruções ou processos.

Setas conectam essas formas, delineando o fluxo lógico do programa. Decisões são frequentemente representadas por losangos, permitindo bifurcações no caminho conforme as condições serão abordadas mais à frente nesta disciplina.

A natureza visual do fluxograma simplifica a compreensão, mesmo para quem não é proficiente em linguagens de programação. Ao seguir as setas e formas, é possível visualizar de forma intuitiva como o algoritmo trabalha. Essa clareza visual é especialmente valiosa durante o design e a depuração de algoritmos, pois oferece uma visão abrangente da lógica subjacente.

Na Figura 1, a seguir, é apresentado um Fluxograma simples, conhecido também como Diagrama de Blocos dentro dos conteúdos de algoritmos presentes na literatura.

ALGORITMOS E TÉCNICAS DE PROGRAMAÇÃO



Figura 1 | Algoritmo para troca de pneu.

ALGORITMOS E TÉCNICAS DE PROGRAMAÇÃO

Em suma, o fluxograma é uma ferramenta essencial que pinta o caminho gráfico do raciocínio lógico por trás de um algoritmo.

Pseudolinguagem

À medida que nos aprofundamos em nossa exploração, deparamo-nos com a pseudolinguagem, uma forma de expressão que combina elementos da linguagem humana com estruturas de programação.

Essa abordagem busca ser uma ponte entre a compreensão intuitiva do algoritmo e sua posterior tradução para uma linguagem de programação específica. A pseudolinguagem oferece uma transição suave do conceitual para o prático, preparando o terreno para a codificação efetiva (MANZANO; OLIVEIRA, 2012).

Podemos dizer, também, que a pseudolinguagem é uma espécie de "linguagem de programação", porém ela não é considerada uma linguagem de programação por não ser utilizada profissionalmente. Neste sentido, ela é utilizada academicamente para o aprendizado, por ser de fácil compreensão. O "Portugol" é uma pseudolinguagem que permite o desenvolvimento de algoritmos estruturados em português de forma relativamente mais simples e intuitiva, o que já não acontece com uma linguagem de programação verdadeira.

Na Figura 2, apresentamos um exemplo simples na pseudolinguagem Portugol exemplificando como se programa nesta pseudolinguagem, ou seja, utilizando a forma de escrita: português estruturado.

ALGORITMOS E TÉCNICAS DE PROGRAMAÇÃO

```
1  Algoritmo ExemploPortugol
2  Var
3      caractere nome
4
5  Inicio
6      Escreva("Forneça seu nome...:")
7      Leia(nome)
8
9      Escreva("Seu nome é.....:", nome)
10
11     Escreva("Fim do Programa!")
12 Fim
```

Figura 2 | Pseudolínguagem "Portugol".

Essa representação em pseudolínguagem em Portugol oferece uma visão clara e legível do algoritmo, facilitando o entendimento das etapas lógicas envolvidas na entrada e saída de dados.

Siga em Frente...

Aplicação de algoritmos

Os algoritmos, embora muitas vezes associados à programação de computadores, são na verdade conjuntos de instruções lógicas utilizadas para resolver problemas e realizar tarefas específicas.

A aplicação de algoritmos permeia nosso cotidiano, sendo uma ferramenta fundamental em diversas áreas. Vamos explorar alguns contextos em que a lógica algorítmica desempenha um papel essencial.

ALGORITMOS E TÉCNICAS DE PROGRAMAÇÃO

- **Gerenciamento de listas e tarefas:** em nossas rotinas diárias, frequentemente utilizamos algoritmos para organizar tarefas e listas. Imagine um algoritmo simples que classifica automaticamente suas tarefas em ordem de prioridade, otimizando sua eficiência diária.
- **Sistemas de navegação GPS:** ao inserir um destino no GPS, o sistema utiliza algoritmos para calcular a rota mais rápida ou eficiente, considerando variáveis como distância, tráfego e preferências do usuário.
- **Redes sociais e recomendações:** plataformas de redes sociais aplicam algoritmos para personalizar *feeds*, exibindo conteúdo com base em interesses e interações anteriores. Algoritmos de recomendação também são usados para sugestões de amigos, vídeos ou produtos.
- **Processamento de transações financeiras:** no setor financeiro, algoritmos são indispensáveis para realizar operações como verificação de transações, detecção de fraudes e análise de padrões de gastos.
- **Medicina e diagnósticos:** algoritmos são empregados em sistemas de apoio ao diagnóstico médico, analisando dados clínicos para identificar padrões e sugerir possíveis tratamentos.
- **Filtros de spam em e-mails:** algoritmos são utilizados para analisar o conteúdo de e-mails e identificar padrões associados a mensagens indesejadas, contribuindo para a eficácia dos filtros de spam.
- **Classificação de imagens em aplicações fotográficas:** aplicações de reconhecimento de imagem usam algoritmos para classificar e organizar fotos com base em critérios como localização, rostos reconhecidos e data de captura.
- **Otimização de rotas de entrega:** empresas de logística utilizam algoritmos para otimizar rotas de entrega, minimizando custos e tempo, e garantindo uma distribuição eficiente.
- **Aprendizado de máquina e inteligência artificial:** algoritmos de aprendizado de máquina são considerados a espinha dorsal da inteligência artificial, capacitando sistemas a aprender e tomar decisões com base em dados.

A aplicação de algoritmos vai além da programação, alcançando áreas diversas de nossas vidas. Essas instruções lógicas fornecem soluções eficientes para uma ampla gama de desafios, desde a organização de tarefas diárias até o desenvolvimento de tecnologias avançadas. Ao compreender a importância dos algoritmos, somos capacitados a utilizá-los de maneira consciente e inovadora, contribuindo para uma sociedade cada vez mais conectada e eficiente.

Vamos Exercitar?

Descrição da situação-problema:

Seu desafio é criar um conjunto de passos simples para preparar uma xícara de café. Não precisamos de código por enquanto, apenas uma descrição narrativa do processo.

Resolução da situação-problema:

Verificação da Cafeteira:

ALGORITMOS E TÉCNICAS DE PROGRAMAÇÃO

- Antes de tudo, verifique se a cafeteira está pronta para uso.

Ligar a Cafeteira:

- Se a cafeteira estiver desligada, ligue-a.

Abastecer Água:

- Confira o nível de água na cafeteira.
- Se estiver baixo, adicione água até atingir a quantidade desejada.

Adicionar Café:

- Verifique a quantidade de café disponível.
- Se estiver acabando, adicione a quantidade necessária.

Preparar Café:

- Pressione o botão de preparo na cafeteira.
- Aguarde o processo de preparação ser concluído.

Servir Café:

- Com cuidado, despeje o café na xícara.

Apreciar:

- Por fim, sente-se, relaxe e aprecie sua xícara de café!

Esses passos representam um algoritmo simples para preparar café. Descrever o processo dessa maneira ajuda a entender a lógica por trás da tarefa antes de entrar em detalhes de programação. Este é apenas o início de sua jornada nos algoritmos.

Saiba mais

Para saber mais sobre os conceitos e Introdução aos algoritmos no que diz respeito à formação de ideias para compreensão desse conteúdo, acesse em sua Biblioteca Virtual o livro: [Estudo Dirigido de Algoritmos](#), Capítulo 1, páginas 15 a 24.

Referências

ALGORITMOS E TÉCNICAS DE PROGRAMAÇÃO

CORMEN, T. **Algoritmos – Teoria e Prática**. 3.ed. Rio de Janeiro: LTC, 2022.

CORMEN, T. **Desmistificando Algoritmos**. Grupo GEN, 2013.

FERREIRA, A. B. H. **Novo Aurélio Século XXI**: o dicionário da língua portuguesa. 3 ed. totalmente rev. e ampl. Rio de Janeiro: Nova Fronteira, 1999.

MANZANO, J. A. N. G.; OLIVEIRA, J. F. **Estudo Dirigido de Algoritmos**. [S. I.]: Editora Saraiva, 2012.

MENÉNDEZ, A. **Simplificando Algoritmos**. Grupo GEN, 2023. E-book.

SERPA, M. S.; RODRIGUES, T. N.; ALVES, Í. C.; *et al.* **Análise de Algoritmos**. Grupo A, 2021.

Aula 2

Conceitos Básicos de Linguagens de Programação

Conceitos básicos de linguagens de programação

Este conteúdo é um vídeo!



Para assistir este conteúdo é necessário que você acesse o AVA pelo computador ou pelo aplicativo. Você pode baixar os vídeos direto no aplicativo para assistir mesmo sem conexão à internet.

Dica para você

Aproveite o acesso para baixar os slides do vídeo, isso pode deixar sua aprendizagem ainda mais completa.

Olá, estudante! Nesta videoaula, você verá os caminhos da sintaxe e semântica de linguagens de programação. Aprofundaremos ainda mais, explorando tipos de dados e desvendando os conceitos fundamentais das estruturas de controle básicas.

Viaje pelo universo dos dados, conhecendo suas variedades e entendendo como escolher o tipo certo de dado, pode potencializar suas soluções.

Você verá sobre estruturas de controle, aprendendo a dar direção e eficiência aos seus algoritmos.

Esteja pronto para essa imersão e não perca a oportunidade de aprimorar seus conhecimentos. Sua participação é indispensável para tornar essa experiência ainda mais enriquecedora.

ALGORITMOS E TÉCNICAS DE PROGRAMAÇÃO

Ponto de Partida

Olá, estudante! Nesta Aula, iniciaremos uma jornada conjunta na exploração dos conceitos básicos de linguagens de programação. Ao longo dessa Aula, vamos desvendar os mistérios para à criação de soluções eficazes em forma de algoritmos computacionais.

Serão apresentados conteúdos que o conduzirão ao vasto universo da programação. Vamos abordar a sintaxe e semântica de linguagens de programação e os possíveis tipos de dados utilizados no desenvolvimento de soluções algorítmicas.

Ao final desta Aula, você vai compreender alguns conceitos de estruturas de controle que serão utilizadas dentro do código-fonte.

Ligado ao desenvolvimento do raciocínio lógico, encontra-se a necessidade de estudar os elementos responsáveis por organizar esse raciocínio transformando em um algoritmo que solucione problemas propostos.

Diante disso, será apresentado a criação de um algoritmo para gerenciar informações sobre produtos comercializados por uma loja de conveniência.

Vamos lá?

Vamos Começar!

O objetivo desta aula é apresentar os conceitos básicos de linguagens de programação, compreendendo aspectos necessários que moldam a construção de algoritmos corretos e eficientes.

Ao compreender os conceitos da linguagem de programação, estaremos capacitados a expressar nossas ideias de maneira precisa e coerente em códigos comprehensíveis.

Ao longo desta Aula, exploraremos os conteúdos que serão abordados com exemplos práticos e contextualizados. Prepare-se para absorver conhecimentos que não apenas enriquecerão seu repertório como programador, mas também abrirão as portas para a criação de soluções mais sofisticadas e eficazes.

Vamos desbravar os alicerces da programação, capacitando-o a construir algoritmos robustos e funcionais. Este é apenas o começo de uma jornada fascinante!

Sintaxe e semântica de linguagens de programação

ALGORITMOS E TÉCNICAS DE PROGRAMAÇÃO

A linguagem de programação é o veículo que nos permite comunicar instruções aos computadores, e seu entendimento é essencial para quem busca se aventurar no mundo da programação. Neste contexto, a sintaxe e semântica são pilares que fundamentam a construção de códigos coerentes e funcionais.

Iniciaremos nossa jornada mergulhando nesses conceitos de linguagens de programação. Imagine esses elementos como as regras gramaticais e o significado de uma linguagem, semelhantes à forma como palavras e frases constroem o entendimento em nossa comunicação cotidiana. Compreender essas estruturas é fundamental para expressar nossas ideias de forma clara e coesa no universo da programação.

Definições de sintaxe e semântica

A sintaxe refere-se às regras gramaticais e estruturais de uma linguagem de programação. É como a gramática de um idioma, determinando como os elementos devem ser organizados para criar instruções válidas.

Por exemplo, no português escrevemos "Sujeito + Verbo + Objeto," do mesmo modo, em uma linguagem de programação, seguimos regras específicas para criar comandos comprehensíveis pela máquina (MENÉNZ, 2023).

A semântica, por sua vez, está relacionada ao significado das instruções. É como garantimos que, além de seguir as regras gramaticais, nossas instruções têm o significado desejado. Usando o mesmo exemplo, se em português dissermos "Planta + Água," sabemos que estamos falando sobre regar uma planta. Na programação, é importante que nosso código faça sentido e realize a ação esperada (MENÉNZ, 2023).

Analizando a sintaxe no idioma português, percebemos que seguimos estruturas específicas para formar frases. Verbos têm conjugações específicas para cada sujeito, e a ordem das palavras impacta o significado da frase. Em programação, a sintaxe segue uma lógica semelhante.

Palavras-chave, variáveis e operadores devem ser dispostos de maneira específica para que o código seja compreendido e executado corretamente.

Falando sobre o Portugol, que será nossa "pseudolínguagem" neste primeiro momento, vale lembrar que ele também é conhecido como português estruturado, é uma pseudolínguagem que possui como base a língua portuguesa.

A sintaxe no Portugol

O Portugol, uma linguagem pseudocódigo utilizada para ensinar programação que simplifica a sintaxe, e permite que o estudante compreenda os conceitos fundamentais antes de enfrentar

ALGORITMOS E TÉCNICAS DE PROGRAMAÇÃO

linguagens mais complexas. Nela, os comandos são escritos de forma mais próxima do idioma humano, facilitando a transição do entendimento gramatical para a lógica de programação.

É por esse motivo que é recomendado primeiro o aprendizado do Portugol, ou seja, uma pseudolínguagem do que ir direto para uma linguagem de programação como C, Java ou Python.

Vamos explorar um exemplo prático na linguagem Portugol para reforçar esses conceitos. Suponha que desejamos criar um algoritmo que some dois números. Em Portugol, poderíamos ter algo como o apresentado na Figura 1.

```
1  ✓ Algoritmo Soma
2  ✓   Var
3      |   num1, num2, resultado: Real
4  ✓   Início
5      |   Escreva("Digite o primeiro número: ")
6      |   Leia(num1)
7
8      |   Escreva("Digite o segundo número: ")
9      |   Leia(num2)
10
11     |   resultado <- num1 + num2
12
13     |   Escreva("A soma é: ", resultado)
14   Fim
```

Figura 1 | Algoritmo "Soma".

Neste exemplo, a sintaxe é clara: temos comandos como "Escreva," "Leia," e o uso de operadores matemáticos. A semântica também é evidente, já que o código realiza a ação esperada, que é somar dois números.

Compreender a sintaxe e semântica é o alicerce para criar programas robustos. À medida que avançamos nos estudos, a clareza na escrita do código se torna essencial. Assim como em um idioma, praticar a construção de frases coesas é crucial para nos expressarmos eficientemente.

ALGORITMOS E TÉCNICAS DE PROGRAMAÇÃO

Para o entendimento completo do exemplo abordado na Figura 1, é necessário além de entender a sintaxe e semântica da escrita, outros conteúdos como por exemplo o conceito de tipos de dados.

Siga em Frente...

Tipos de dados

Os tipos de dados são a base fundamental para representar e manipular informações em um programa. Eles definem o conjunto de valores que uma variável pode armazenar, proporcionando uma riqueza de expressão aos nossos algoritmos.

Vejamos os quatro principais tipos de dados: **inteiros, reais, caracteres e lógicos**, compreendendo suas peculiaridades e explorando como são utilizados na linguagem de programação.

Inteiros

Os dados do tipo inteiro representam números inteiros, ou seja, valores sem parte fracionária. Em muitas linguagens de programação, como também na pseudolínguagem Portugol, os inteiros podem ser positivos ou negativos.

Por exemplo: os valores (0, -15, 4185, -3760) são exemplos de valores classificados como inteiros. Costumamos dizer também que o tipo de dado define qual a informação possível que será trabalhada, ou seja, quando definimos um objeto qualquer como "inteiro", estamos limitando a somente esse tipo de valor. Números sem casas decimais como exemplificado.

Reais

Os dados do tipo real representam números inteiros, e números fracionários, ou seja, valores numéricos de qualquer tipo. Podemos fazer uma comparação dizendo que os tipos reais atendem o que os tipos inteiros atendem e ainda são capazes de trabalhar com números fracionários. Se fosse uma concorrência de competências, os tipos reais certamente levariam a melhor.

Assim, se os tipos reais são uma "melhoria" dos tipos inteiros, pois atendem suas especificações e ainda com a possibilidade de trabalhar com números fracionários, por que usar os tipos inteiros? Vamos ficar somente com os tipos reais, certo? Errado, pois os tipos inteiros utilizam menos memória que os tipos reais.

Portanto, sempre que pudermos utilizar tipos inteiros, devemos fazê-lo. Um exemplo clássico é a idade de uma pessoa. Será geralmente um valor inteiro. Quando este for o caso, use o tipo de

ALGORITMOS E TÉCNICAS DE PROGRAMAÇÃO

dado inteiro.

Exemplos de valores dos tipos reais são: (0, -15.3, 418.6, -7.509). Dentro dos algoritmos é muito comum o processamento de cálculos matemáticos. Quando o seu programa utilizar a operação de divisão e em outros casos de cálculos mais complexos, você vai precisar utilizar tipos reais no seu algoritmo.

Caracteres

Para representar palavras, textos ou símbolos especiais ou ainda números em conjunto com informações textuais, utilizamos o tipo de dado caractere.

Por exemplo, para trabalhar com a informação do endereço de uma pessoa, teremos valores textuais em conjunto com números, "**Rua Tecnologia da Informação, 5000**". Repare que essa informação de um endereço qualquer tem o logradouro, o endereço e o número, ou seja, informações de vários tipos que serão representadas pelo tipo: caractere.

Essa categoria inclui letras, números, símbolos e espaços. A forma de representação sempre será dentro de aspas duplas " ", como representado no parágrafo anterior.

Explicando melhor, se o número **5000** aparecer sozinho, ele pode ser considerado um valor inteiro, mas se aparecer entre as aspas "**5000**", ele passa a ser caractere. Outra restrição para o número 5000, agora sendo tratado como caractere é a impossibilidade de operações matemáticas, pois agora ele não é um valor numérico e sim um valor textual.

Lógicos

O tipo de dado lógico lida com valores de verdadeiro ou falso. Eles são importantes dentro do desenvolvimento de algoritmos mais complexos. Assim, os valores lógicos conseguem trabalhar somente com 2 valores: VERDADEIRO ou FALSO.

Para uma linguagem de programação, trabalhada no idioma inglês, esses valores serão TRUE ou FALSE. Eles são utilizados para armazenar o retorno de uma função, por exemplo.

Em Portugol, a manipulação desses tipos de dados é intuitiva. Podemos realizar operações matemáticas, comparações e concatenações de caracteres de forma simples. A linguagem facilita o entendimento dos conceitos antes de lidar com sintaxes mais complexas.

A seguir, na Tabela 1, é apresentada uma tabela com um exemplo de valores para cada tipo de dado.

Tipo de dado	Exemplo
Inteiro	42

ALGORITMOS E TÉCNICAS DE PROGRAMAÇÃO

Real	3.14
Caractere	"A"
Lógico	Verdadeiro

Tabela 1 | Exemplo tipos de dados.

Essa Tabela apresenta exemplos representativos de cada tipo de dado. Os inteiros e reais são utilizados para números, o caractere para representar letras ou símbolos, e o lógico para valores verdadeiro ou falso. Esses exemplos ajudam a compreender como os diferentes tipos de dados podem ser aplicados em programação.

Dominar os tipos de dados é uma jornada essencial na construção do conhecimento em programação. Cada tipo possui sua função única, e a escolha adequada é crucial para o sucesso do algoritmo.

Execução de um programa de computador

De acordo com Manzano; Oliveira (2012), os algoritmos criados por desenvolvedores devem ser convertidos para uma linguagem de alto nível, como: C, Java, etc. Devemos converter o algoritmo escrito para uma dessas linguagens de alto nível para que eles se tornem executáveis, mas cada linguagem possui um método único para realizar este processo.

De acordo com Manzano; Oliveira (2012), existem três métodos para gerar um código executável: Compiladores, Interpretadores e Tradutores.

Uma advogada, um estudante, um professor e diversos outros profissionais que utilizam computadores em suas atividades normalmente recorrem a uma ferramenta chamada processador de textos. Contadores, engenheiros, economistas, físicos, químicos, matemáticos e outros profissionais que lidam com cálculos fazem uso de uma ferramenta denominada planilha eletrônica ou por meio da nuvem utilizando serviços de editores compartilhados.

Já publicitários, desenhistas e profissionais de comunicação utilizam ferramentas gráficas, como programas de apresentação ou editores gráficos de desenho e fotografia. Na área de desenvolvimento de sistemas, os desenvolvedores de software também têm suas próprias ferramentas de trabalho, como editores de texto, compiladores e tradutores.

Como citado, ao concluir o projeto de um programa de computador, é necessário transformá-lo em software. Para isso, o projeto definido é traduzido para uma linguagem de programação formal, que é executada em um computador. Essa tarefa envolve escrever o código do programa em uma ferramenta de edição de textos e, em seguida, submeter o programa a ferramentas de tradução, interpretação e compilação, conforme necessário.

ALGORITMOS E TÉCNICAS DE PROGRAMAÇÃO

O editor de texto, ferramenta utilizada para editar o texto de um programa, é um programa simples que permite escrever o código do programa e, basicamente, gravá-lo. Essa ferramenta possui recursos essenciais, como a capacidade de copiar e colar blocos de texto, salvar e recuperar arquivos e imprimi-los.

Exemplos de editores de texto incluem o EDIT no MS-DOS, o Notepad do Windows, conhecido como bloco de notas, e o VI (vi-ai) no Linux e UNIX. Não é necessário que um editor de texto ofereça mais recursos do que esses, pois, quando isso acontece, o programa deixa de ser um editor e passa a ser um processador de texto, como o "WordPad" ou "Microsoft Word" no Windows ou outros similares em outros sistemas operacionais.

As ferramentas de tradução são programas que possibilitam a conversão de um código em uma linguagem formal para outra linguagem também formal. Suponha que um programador, embora seja proficiente em programação e tenha uma lógica excelente, só consiga escrever programas na linguagem Java, e necessite entregar um programa-fonte em Python. O desafio é que ele não tem conhecimento da linguagem Python.

Nessa situação, ele pode recorrer a uma ferramenta de tradução. O programador escreve o programa-fonte em Java, e a ferramenta de tradução reescreve o código na linguagem Python, gerando um programa-fonte como se o próprio programador tivesse escrito. Há diversos tipos de tradutores disponíveis, tanto ferramentas de distribuição gratuita, como o programa p2c, quanto ferramentas comerciais.

As ferramentas de interpretação são programas que executam um programa-fonte diretamente na memória principal do computador, sem a necessidade de ser executado no processador central da máquina. Geralmente, esse tipo de ferramenta permite uma execução rápida dos programas. Diversas linguagens de programação, como Basic, Perl, Python, Forth, JavaScript e LOGO, adotam a interpretação como método de execução.

Já as ferramentas de compilação são programas que convertem um programa-fonte escrito em uma linguagem de alto nível para uma linguagem de baixo nível (linguagem de máquina).

Durante essa tradução, o programa-fonte transforma-se em um programa-objeto compatível com o processador em uso e, posteriormente, ocorre a ligação com as rotinas de execução do sistema operacional, transformando-o em um código executável. Embora os programas compilados possam apresentar um desempenho ligeiramente mais lento se comparados aos programas interpretados, eles asseguram a inacessibilidade ao código-fonte.

A escolha entre ferramentas de tradução, interpretação ou compilação, bem como a linguagem de programação a ser utilizada, depende de uma análise criteriosa e decisões a serem tomadas pela equipe de desenvolvimento alinhada com a gestão da empresa.

Em conclusão, as ferramentas de tradução, interpretação e compilação desempenham papéis únicos no ciclo de vida do desenvolvimento de software. A escolha entre essas abordagens

ALGORITMOS E TÉCNICAS DE PROGRAMAÇÃO

depende das necessidades específicas de um projeto, considerando fatores como desempenho, acessibilidade ao código-fonte e as características da linguagem de programação.

As ferramentas de tradução oferecem a flexibilidade de converter códigos entre linguagens formais, como exemplificado pelo programa p2c. Já as ferramentas de interpretação permitem uma execução rápida dos programas diretamente na memória, mas podem expor o código-fonte.

Por fim, as ferramentas de compilação garantem a transformação do código-fonte em um programa executável, proporcionando maior segurança, embora com um potencial leve impacto no desempenho. A compreensão dessas ferramentas é essencial para os desenvolvedores, pois influencia diretamente na eficiência e na segurança das soluções computacionais que eles desenvolvem.

Portanto, a compreensão da sintaxe e semântica de linguagens de programação é essencial para construir códigos claros e funcionais. A exploração dos tipos de dados, incluindo inteiros, reais, caracteres e lógicos, proporciona ao programador a capacidade de manipular informações de forma eficiente.

Além disso, entendemos o processo de execução de um programa de computador, compreendendo as etapas de tradução, interpretação e compilação.

Todos esses conhecimentos são necessários para qualquer programador, constituindo os alicerces para a resolução de problemas complexos e o desenvolvimento de aplicações robustas e de alta performance.

Vamos Exercitar?

Descrição da situação-problema:

Em um ambiente de controle de estoque de uma loja de conveniência, surge a necessidade de desenvolver um algoritmo para gerenciar informações sobre os produtos disponíveis. O objetivo é garantir maior eficiência nas operações relacionadas ao estoque, como consulta e atualização de dados.

Descrição da Resolução:

Para solucionar essa demanda, é necessário criar um programa que conte cole diferentes tipos de dados para representar as características dos produtos. Cada produto deve ser identificado por um código (íntero), ter um preço unitário associado (real), uma descrição textual (caractere) e um indicador de disponibilidade em estoque (lógico).

Resolução da situação-problema:

ALGORITMOS E TÉCNICAS DE PROGRAMAÇÃO

Definição dos tipos de dados:

- Código: Inteiro.
- Preço Unitário: Real.
- Descrição: Caractere.
- Disponibilidade em Estoque: Lógico.

Registro de produtos:

- Cada produto será registrado com um código único, preço, descrição e disponibilidade inicial.

Consulta de produtos:

- O algoritmo deve permitir a consulta de informações de um produto específico, fornecendo seus detalhes.

Atualização de estoque:

- Operações para aumentar ou diminuir a quantidade de um produto em estoque, atualizando o indicador de disponibilidade.

Utilização do Algoritmo:

- O algoritmo será implementado em um sistema que permitirá aos funcionários gerenciar o estoque da loja de maneira eficiente.

Essa situação-problema introduz a necessidade de compreender e manipular tipos de dados, a saber: (inteiros, reais, caracteres e lógicos) para criar um algoritmo funcional no contexto do controle de estoque.

O desenvolvimento e teste do algoritmo proporcionarão *insights* sobre como esses tipos de dados são essenciais na resolução de problemas práticos.

Saiba mais

Para saber mais sobre os conceitos e Introdução aos algoritmos no que diz respeito à formação de ideias para compreensão desse conteúdo, acesse em sua Biblioteca Virtual o livro: [Estudo Dirigido de Algoritmos](#), Capítulo 3 - "Programação com Sequência páginas 38 a 40.

Referências

ALGORITMOS E TÉCNICAS DE PROGRAMAÇÃO

CORMEN, T. **Algoritmos – Teoria e Prática**. 3.ed. Rio de Janeiro: LTC, 2022.

CORMEN, T. **Desmistificando Algoritmos**. Grupo GEN, 2013.

FERREIRA, A. B. H. **Novo Aurélio Século XXI**: o dicionário da língua portuguesa. 3 ed. totalmente rev. e ampl. Rio de Janeiro: Nova Fronteira, 1999.

MANZANO, J. A. N. G.; OLIVEIRA, J. F. **Estudo Dirigido de Algoritmos**. [S. I.]: Editora Saraiva, 2012.

MENÉNDEZ, A. **Simplificando Algoritmos**. Grupo GEN, 2023. E-book.

SERPA, M. S.; RODRIGUES, T. N.; ALVES, Í. C.; *et al.* **Análise de Algoritmos**. Grupo A, 2021.

Aula 3

Componentes e Elementos de Linguagem de Programação

Componentes e elementos de linguagem de programação

Este conteúdo é um vídeo!



Para assistir este conteúdo é necessário que você acesse o AVA pelo computador ou pelo aplicativo. Você pode baixar os vídeos direto no aplicativo para assistir mesmo sem conexão à internet.

Dica para você

Aproveite o acesso para baixar os slides do vídeo, isso pode deixar sua aprendizagem ainda mais completa.

Olá, estudante! Nesta videoaula, abordaremos temas como atribuições de valores, estrutura de um programa de computador e alguns fundamentos de variáveis e constantes.

Durante a Aula, desvendaremos os segredos por trás das operações básicas da matemática, e como essas operações são atribuídas a variáveis criando algoritmos consistentes. Além disso, exploraremos a estrutura essencial dos algoritmos e programas, proporcionando uma visão clara de como organizar suas instruções para alcançar soluções eficientes.

Estamos ansiosos para tê-lo conosco nesta jornada de aprendizado. Prepare-se para expandir seus horizontes na ciência da computação! Não perca essa oportunidade e confira a videoaula. Até lá!

ALGORITMOS E TÉCNICAS DE PROGRAMAÇÃO

Ponto de Partida

Olá, estudante!

Vamos abordar os fundamentos essenciais dos conteúdos dessa aula, para que no decorrer desta jornada você fique mais próximo do objetivo de dominar as técnicas dos algoritmos e da lógica de programação.

Ao abordar a "Introdução às operações e atribuições", veremos os princípios das ações executadas pelos algoritmos. Compreender como realizar operações e atribuições é essencial para construir lógicas eficientes na resolução de problemas computacionais.

A "Estrutura dos algoritmos e programas" será outro ponto focal. Vamos analisar como os algoritmos são organizados e estruturados para criar programas coesos e compreensíveis.

Além disso, exploraremos os "Fundamentos de variáveis e constantes". Variáveis e constantes são elementos-chave na programação, permitindo armazenar e manipular dados. Entender como utilizá-las corretamente é essencial para a criação de algoritmos eficazes.

Atualmente, o desenvolvimento de sistemas e aplicativos ligados a programação de computadores está muito presente na sociedade. Assim, o desenvolvimento do raciocínio lógico é fundamental para que o profissional se desenvolva em qualquer área do conhecimento que deseja atuar.

Diante disso, será apresentado o esboço de um algoritmo para automatizar uma cortina em uma residência por meio de passos executados em uma sequência lógica que tornarão possível esta atividade.

Convido você a iniciar este conteúdo para continuar sua jornada no universo de aprendizado da programação. Vamos lá?

Vamos Começar!

Olá, estudante! Nesta Aula, faremos uma introdução a operações e atribuições, explorando como manipular e modificar dados, realizar cálculos e executar ações fundamentais para a resolução de problemas computacionais. Compreenderemos como essas operações são essenciais na construção de algoritmos que não apenas funcionam, mas que também são eficientes e elegantes.

Entenderemos como criar fluxos de execução que transformam problemas complexos em passos claros e compreensíveis, uma habilidade crucial para qualquer programador que quer entrar no mercado de trabalho.

ALGORITMOS E TÉCNICAS DE PROGRAMAÇÃO

Não menos importante, mergulharemos nos fundamentos de variáveis e constantes, elementos-chave que conferem dinamismo e adaptabilidade aos nossos algoritmos. Compreenderemos como esses componentes são essenciais para armazenar e manipular dados, possibilitando a criação de programas com esses recursos.

Fundamentos de variáveis e constantes

Na vasta paisagem da programação, os fundamentos de variáveis e constantes se destacam como pilares essenciais, conferindo flexibilidade e controle à manipulação de dados nos algoritmos.

Vejamos, então, o que são variáveis, e como utilizá-las de maneira eficaz no contexto do Portugol. As variáveis serão utilizadas em todas as linguagens de programação, porém neste momento vamos nos limitar em nossa pseudolínguagem, o Portugol.

Em Portugol, uma variável é um espaço de armazenamento identificado por um nome simbólico, que representa um valor ou informação. Ao contrário das constantes, as variáveis podem ter seu valor alterado durante a execução do programa. Elas são como gavetas na memória, onde podemos guardar e modificar dados conforme necessário metaforicamente falando.

A memória do dispositivo que estamos utilizando funciona como se fossem repositórios, caixas onde podemos colocar as informações dentro delas. Elas estão relacionadas aos tipos de dados, onde em cada gaveta podemos colocar somente valores do mesmo tipo.

Pense, por exemplo, em duas gavetas de meias e camisetas. Na gaveta de meias você não pode colocar camisetas, somente meias. Da mesma forma quando você cria uma variável você define que dentro dela você só vai aceitar valores numéricos, ou seja, se você quiser armazenar o nome de uma pessoa isso não será possível, pois o nome não é um valor numérico, logo você precisa criar outra gaveta, ou seja, outra variável (MENÉNDEZ, 2023).

As variáveis têm características essenciais, como o tipo de dado que podem armazenar, seja ele inteiro, real, caractere ou lógico. Além disso, possuem um escopo, determinando em que no programa elas podem ser acessadas, e uma sintaxe específica para declaração, indicando ao Portugol a reserva de espaço na memória (MENÉNDEZ, 2023).

Variáveis são indispensáveis para armazenar informações dinâmicas, como resultados intermediários, dados do usuário ou estados temporários. Podem conter valores que variam ao longo da execução do programa, proporcionando adaptabilidade.

No Portugol, seus valores possíveis são diretamente relacionados ao tipo de dado especificado durante a declaração como já citado nesta aula. Se você cria uma variável do tipo caractere, significa que vai poder armazenar valores textuais dentro da mesma (MENÉNDEZ, 2023).

ALGORITMOS E TÉCNICAS DE PROGRAMAÇÃO

Quando uma variável é criada, o Portugol aloca um espaço na memória do computador para armazenar seu valor. É fundamental inicializar uma variável antes de utilizá-la para evitar comportamentos inesperados, pois seu valor inicial pode ser indefinido se não especificado explicitamente.

Desse modo, a inicialização é o ato de atribuir um valor inicial à variável no momento da declaração. Isso pode evitar que exista "sujeita" no local onde a variável está utilizando na memória, visto que a memória pode ser compartilhada por outros programas e aplicativos na máquina que estão sendo executados concomitantemente. Assim, quase todos os algoritmos existentes utilizam um conjunto de variáveis no seu interior, por isso é muito importante absorver bem este conceito (MENÉNZ, 2023).

Outra questão é compreender que a variável que está no algoritmo é passível de alteração, ou seja, o seu conteúdo é volátil, incerto e totalmente alterável (MENÉNZ, 2023).

A exemplo disso, imaginemos que cada variável que utilizamos é constituída de três partes: **nome, tipo e conteúdo**.

O nome é dado quando declaramos a variável, assim como o tipo de informação que a variável vai trabalhar daqui pra frente. O dado que será colocado dentro da variável, virá de um **comando de atribuição** dentro do algoritmo ou de uma **leitura** da própria variável. Importante também ter este conceito em mente. Outros conceitos serão abordados posteriormente. Neste momento o que precisamos compreender são essas três partes que constituem uma variável.

Para exemplificar, na Tabela 1 é apresentada um esquema de variável apresentando o seu "MÓDELO", constituído por essas três partes e um exemplo de uma variável já "CRIADA".

Modelo de uma variável	Uma variável qualquer
Nome	Idade
Tipo de Dado	Inteiro
Valor	19

Tabela 1 | Esquema de uma variável.

Neste universo intrigante de variáveis e constantes em Portugol, compreendemos que elas não são apenas recipientes de dados, mas componentes dinâmicos que dão vida e flexibilidade aos nossos algoritmos.

Ao entender suas características, utilização e importância da inicialização, capacitamos nossos programas a lidarem de forma eficaz com a complexidade dos dados e das operações. Este

ALGORITMOS E TÉCNICAS DE PROGRAMAÇÃO

conhecimento é essencial para qualquer programador que busca construir soluções robustas e adaptáveis.

Introdução as operações e atribuições

Na jornada da programação, a habilidade de realizar operações e atribuições é a essência da manipulação de dados em algoritmos. Vamos nos aprofundar nesse território fundamental, começando pela compreensão dos operadores de atribuição no contexto do Portugol.

Operadores de atribuição

Em Portugol, os operadores de atribuição são fundamentais para a manipulação de variáveis. O mais comum é o operador `=` (seta pra esquerda), utilizado para atribuir um valor a uma variável. Por exemplo, `idade = 25` atribui o valor 25 à variável idade. Esses operadores são como ferramentas que permitem moldar e modificar o comportamento do programa.

Os operadores de atribuição não apenas designam valores a variáveis, mas também podem ser combinados com outros operadores, como os operadores aritméticos por exemplo: `(+, -, *, /)`, proporcionando uma forma concisa de realizar operações e atribuições simultaneamente. Por exemplo, `idade = idade + 1` aumenta o valor da variável idade em 1.

Esses operadores são amplamente utilizados em situações em que é necessário atualizar, calcular ou modificar o valor de variáveis durante a execução do programa. Seja em um simples incremento numérico ou em operações mais complexas, os operadores de atribuição são ferramentas indispensáveis para o desenvolvedor.

Alteração de valores

A alteração de valores de uma variável pode ocorrer de duas formas em Portugol. A primeira é por meio do comando de leitura, onde o usuário fornece um valor que é então atribuído à variável. A segunda forma é pelo operador de atribuição, onde podemos diretamente especificar o valor a ser atribuído à variável.

Exemplificando melhor, o comando de leitura de dados é utilizado toda vez que é necessário capturar informações externas ao programa. Por exemplo: um algoritmo que solicite ao usuário sua senha. Este valor é uma entrada de dados externo que será utilizado na verificação dentro do algoritmo. A Figura 2 apresenta um código em Portugol mostrando este processo. Observe.

ALGORITMOS E TÉCNICAS DE PROGRAMAÇÃO

```
1  Algoritmo Senha
2      Var
3          nome, senha: caractere
4      Início
5          Escreva("Digite o seu nome...: ")
6          Leia(nome)
7
8          Escreva("Entre com sua senha.: ")
9          Leia(senha)
10
11         Escreva("Fim do Programa!!!")
12     Fim
```

Figura 2 | Comando de leitura de dados.

Como pode ser observado na Figura 2, a linha 9 do algoritmo "Senha", o comando `Leia()` faz a gravação do valor digitado quando o algoritmo está sendo executado na variável "senha". A partir desse momento o valor da variável é alterado.

A Figura 3, também apresenta a outra forma de mudança do valor da variável por meio do operador de atribuição, observe.

ALGORITMOS E TÉCNICAS DE PROGRAMAÇÃO

```

1   Algoritmo Senha
2       Var
3           nome, senha: caractere
4       Início
5           Escreva("Digite o seu nome...: ")
6           Leia(nome)
7
8           Leia <- "****45JW"
9
10          Escreva("Fim do Programa!!!")
11      Fim

```

Figura 3 | Operador de atribuição.

Percebemos que o algoritmo foi alterado para que o valor da variável senha seja alterado pelo operador de atribuição <- (seta para esquerda).

Ao entender como manipular variáveis por meio de operadores de atribuição, o programador comprehende umas das funções mais utilizadas no desenvolvimento de algoritmos. A capacidade de manipular variáveis por meio de atribuições não é apenas um detalhe técnico; é a chave para desbloquear o potencial de criação e inovação no mundo da programação.

Siga em Frente...

Estrutura dos algoritmos e programas

No mundo da programação, a estrutura dos algoritmos e programas é um conceito fundamental que permeia toda a disciplina. Assim como a arquitetura de um edifício proporciona a base para sua funcionalidade, a estrutura dos algoritmos determina como as instruções são organizadas e executadas dentro de um programa de computador.

Esta é a essência que dá forma à lógica por trás do código, proporcionando uma compreensão clara e eficiente do funcionamento do software.

ALGORITMOS E TÉCNICAS DE PROGRAMAÇÃO

Explorando o universo da programação por meio do Portugol, uma linguagem simplificada e didática, somos introduzidos às estruturas fundamentais que constituem qualquer algoritmo.

A sequência, nossa primeira estaca, organiza as instruções em uma ordem linear, criando uma narrativa coerente. Em paralelo, o comando de atribuição entra em cena, permitindo-nos definir valores a variáveis e, assim, moldar dinamicamente o comportamento do nosso algoritmo.

Construindo com comandos simples, neste estágio inicial, optamos por comandos mais comuns, afastando-nos temporariamente das complexidades das estruturas condicionais, de repetição ou funções. Essa abordagem minimalista nos fornece uma visão clara da essência da programação, permitindo-nos focar na estruturação básica dos nossos algoritmos.

Portanto, cada linha de código, cada sequência de comandos, torna-se uma pedra preciosa na construção do conhecimento, preparando-nos para desafios mais avançados que virão à medida que avançamos nessa incrível jornada.

A estrutura dos algoritmos e programas desempenha um papel fundamental na programação, determinando como as instruções são organizadas e executadas. Imaginemos essa estrutura como a arquitetura de uma construção, onde cada elemento tem um lugar específico e uma função clara. No contexto da programação, a organização adequada é crucial para garantir que o código seja comprehensível e eficiente.

Dentro do Portugol, as estruturas fundamentais incluem a sequência, que organiza as instruções em uma ordem linear, e o comando de atribuição, que define valores a variáveis. Essa simplicidade é essencial para construir algoritmos básicos sem a complexidade de estruturas condicionais, de repetição ou funções.

A Figura 4 apresenta o exemplo em Portugol. Observe.

ALGORITMOS E TÉCNICAS DE PROGRAMAÇÃO

```
1  algoritmo "Estrutura_Basica"
2  ∵ var
3  |   numero: inteiro
4  ∵ inicio
5  |   // Sequência de comandos
6  |   escreva("Digite um número: ")
7  |   leia(numero)
8
9  |   // Comando de atribuição
10 |   numero <- numero + 1
11 |
12 |   // Saída
13 |   escreva("O número incrementado é: ", numero)
14
15 fimalgoritmo
```

Figura 4 | Estrutura em Portugol.

A Figura 4 aborda uma estrutura simples em Portugol contendo a definição do nome do algoritmo, uma variável do tipo inteiro declarada, e o algoritmo solicitando o valor ao usuário e incrementando o valor lido somado de mais um valor fixo (1). Ele apresenta uma informação com o valor da variável.

A Figura 5 apresenta o exemplo em Java. Observe que este algoritmo solicita primeiro o valor para o usuário e somente na próxima linha cria a variável já atribuindo o valor lido para a mesma.

ALGORITMOS E TÉCNICAS DE PROGRAMAÇÃO

```

1  / Estrutura sequencial e comando de atribuição em Java
2  public class EstruturaBasica {
3      public static void main(String[] args) {
4          // Sequência de comandos
5          System.out.print("Digite um número: ");
6
7          // Comando de atribuição
8          int numero = Integer.parseInt(System.console().readLine());
9
10         // Saída
11         System.out.println("O número incrementado é: " + (numero + 1));
12     }
13 }
```

Figura 5 | Estrutura em Java.

A Figura 6 apresenta o exemplo em Python. Observamos que este algoritmo solicita o valor de um número ao usuário ao mesmo tempo que cria a variável que armazena o valor lido. Logo após, incrementa-se à variável o valor de 1 e posteriormente é exibida a mensagem do valor da variável na tela do programa.

```

1  # Estrutura sequencial e comando de atribuição em Python
2  # Sequência de comandos
3  numero = int(input("Digite um número: "))
4
5  # Comando de atribuição
6  numero = numero + 1
7
8  # Saída
9  print("O número incrementado é:", numero)
10
```

Figura 6 | Estrutura em Python.

Em conclusão, mesmo algoritmos simples e estruturas básicas são essenciais para compreender os princípios fundamentais da programação. O aprendizado dessa base permite a você, estudante, desenvolver habilidades que serão aplicadas em algoritmos mais complexos, tornando-se a fundação para um aprendizado continuado.

ALGORITMOS E TÉCNICAS DE PROGRAMAÇÃO

Vamos Exercitar?

Descrição da situação-problema:

Em um projeto de automação residencial, surge a necessidade de criar um programa para controlar a abertura e fechamento automático de cortinas em uma casa. O objetivo é facilitar o gerenciamento das cortinas, permitindo que os usuários programem horários específicos para que elas abram ou fechem automaticamente. O desenvolvimento desse sistema de automação será realizado utilizando a pseudolínguagem Portugol.

Resolução da situação-problema:

- Para solucionar esse desafio, inicie a construção de uma estrutura lógica que represente as cortinas automatizadas. Crie um algoritmo em Portugol para definir os atributos essenciais, como o estado das cortinas (abertas ou fechadas), horários programados para abertura e fechamento, entre outros.
- Implemente o processo de abrir e fechar as cortinas, garantindo que o estado seja devidamente atualizado conforme a programação estabelecida pelos usuários.
- Após a aplicação concluída, compile o código em Portugol e realize testes em diferentes cenários para assegurar o correto funcionamento do programa.

Ao finalizar a aplicação, é possível perceber como a estrutura do algoritmo e o design simples do programa permitem a automação eficiente de dispositivos em uma residência. A compreensão dos conceitos de estrutura de algoritmos proporciona a criação de soluções práticas para facilitar tarefas cotidianas por meio da programação.

O exemplo da automação de cortina destaca a versatilidade desses conhecimentos na construção de sistemas úteis e acessíveis.

Saiba mais

Para saber mais sobre os componentes e elementos de linguagem de programação para uma melhor compreensão dos conteúdos abordados nesta aula, acesse em sua Biblioteca Virtual o livro [Simplificando Algoritmos](#), Capítulo 5 “Pseudocódigo”, páginas 25 a 47.

Além de aprofundar seus estudos, você vai conhecer um pouco sobre uma ferramenta que você poderá utilizar para programar em Portugol.

Bons estudos!

Referências

ALGORITMOS E TÉCNICAS DE PROGRAMAÇÃO

CORMEN, T. **Algoritmos – Teoria e Prática**. 3.ed. Rio de Janeiro: LTC, 2022.

CORMEN, T. **Desmistificando Algoritmos**. Grupo GEN, 2013.

FERREIRA, A. B. H. **Novo Aurélio Século XXI**: o dicionário da língua portuguesa. 3 ed. totalmente rev. e ampl. Rio de Janeiro: Nova Fronteira, 1999.

MANZANO, J. A. N. G.; OLIVEIRA, J. F. **Estudo Dirigido de Algoritmos**. [S. I.]: Editora Saraiva, 2012.

MENÉNDEZ, A. **Simplificando Algoritmos**. Grupo GEN, 2023. E-book.

SERPA, M. S.; RODRIGUES, T. N.; ALVES, Í. C.; *et al.* **Análise de Algoritmos**. Grupo A, 2021.

Aula 4

Estruturas de Algoritmos Fundamentais

Estrutura de algoritmos fundamentais

Este conteúdo é um vídeo!



Para assistir este conteúdo é necessário que você acesse o AVA pelo computador ou pelo aplicativo. Você pode baixar os vídeos direto no aplicativo para assistir mesmo sem conexão à internet.

Dica para você

Aproveite o acesso para baixar os slides do vídeo, isso pode deixar sua aprendizagem ainda mais completa.

Olá, estudante! Esta videoaula tem o objetivo de conduzi-lo ao entendimento acerca dos conceitos essenciais que compõem a estrutura de algoritmos fundamentais e de mostrar exemplos práticos que fazem parte da realidade em que estamos inseridos, fazendo com que a compreensão referente ao conteúdo abordado aqui seja efetiva.

Aproveite essa aula para aprimorar seus conhecimentos e expandir sua compreensão sobre o assunto. Junte-se a nós nesta jornada de descobertas e fortaleça suas habilidades na área. Conto com a sua participação!

ALGORITMOS E TÉCNICAS DE PROGRAMAÇÃO

Ponto de Partida

Olá, estudante! Bem-vindo a esta jornada de aprendizado que estamos prestes a iniciar juntos. É uma honra e um privilégio tê-lo conosco nesta busca pelo conhecimento.

Ao longo desta Aula, vamos mergulhar nas estruturas que servem como alicerce de qualquer programa de computador eficiente e robusto, em que cada tópico que abordaremos desempenha um papel necessário no desenvolvimento de algoritmos eficazes e na resolução de problemas.

Lembre-se de que o conhecimento adquirido aqui não se limita às linhas de código ou aos diagramas de fluxo. Estamos construindo uma base sólida para sua jornada na ciência da computação, uma base que não apenas o equipará com habilidades técnicas valiosas, mas também cultivará sua capacidade de pensamento crítico, resolução de problemas e criatividade.

Portanto, será apresentado um algoritmo para auxiliar no controle de estoque de uma loja de produtos eletrônicos. O algoritmo será desenvolvido em Portugol.

Então, vamos juntos embarcar nesta jornada, explorando as maravilhas da Estrutura de Algoritmos Fundamentais. Vamos aprender, crescer e nos inspirar mutuamente nesta busca pelo conhecimento.

Vamos Começar!

Exploraremos, inicialmente, os fundamentos essenciais que moldam a estrutura de qualquer algoritmo eficiente e funcional. Ao longo desta Aula, discorreremos sobre os três pilares fundamentais nessa abordagem: a manipulação de variáveis, a lógica de sequência e controle de fluxo e a introdução ao conceito de funções e modularização.

Ao estudarmos sobre essas áreas fundamentais da programação, estaremos construindo uma base sólida para explorarmos conceitos mais avançados e enfrentarmos desafios cada vez maiores na área que engloba a ciência da computação. Dessa forma exploraremos juntos os princípios essenciais que nos capacitam a transformar ideias em códigos funcionais e eficientes.

Vamos lá?

Conceitos de manipulação de variáveis

A manipulação de variáveis é um conceito central na programação de algoritmos, essencial para quem busca compreender e desenvolver lógica de programação. Em linguagens como Portugol, que é uma linguagem de pseudocódigo utilizada no ensino de programação, entender como as variáveis funcionam é importante para criar algoritmos eficientes e precisos.

ALGORITMOS E TÉCNICAS DE PROGRAMAÇÃO

Variáveis são espaços na memória reservados para armazenar valores que podem ser modificados durante a execução de um programa. Cada variável possui um nome único que a identifica e um tipo que define que tipo de dado ela pode armazenar, como números inteiros, decimais, caracteres, entre outros.

As variáveis possuem algumas características principais, que são:

1. **Nome:** as variáveis têm nomes que as identificam no código. Esses nomes devem seguir regras específicas, como começar com uma letra e não conter espaços.
2. **Tipo:** cada variável é associada a um tipo de dado, indicando o que ela pode armazenar. Por exemplo, uma variável do tipo inteiro só pode conter números inteiros.
3. **Valor inicial:** uma variável pode ser inicializada com um valor específico durante a sua declaração ou mais tarde, durante a execução do programa.

A manipulação de variáveis em Portugol envolve diversas operações que permitem modificar, atribuir e utilizar os valores armazenados. Algumas das operações mais comuns incluem:

1. **Atribuição:** através do operador de atribuição (`<-`), valores podem ser atribuídos a variáveis. Exemplo:

inteiro idade

idade <- 25

2. **Operações matemáticas:** variáveis podem participar de operações matemáticas, como adição, subtração, multiplicação e divisão. Exemplo:

inteiro a, b, resultado

a <- 10

b <- 5

resultado <- a + b

3. **Concatenação de strings:** se uma variável contém uma string, é possível concatená-la com outra utilizando o operador `+`. Exemplo:

caractere nome1, nome2, nomeCompleto

nome1 <- "João"

ALGORITMOS E TÉCNICAS DE PROGRAMAÇÃO

```
nome2 <- "Silva"
```

```
nomeCompleto <- nome1 + " " + nome2
```

4. **Entrada e saída de dados:** variáveis são frequentemente utilizadas para armazenar dados fornecidos pelo usuário (entrada) ou para exibir resultados (saída). Vejamos alguns exemplos:

- **Controle de estoque:** utilizando variáveis para armazenar a quantidade de produtos em estoque e realizando operações de adição e subtração conforme as vendas e reposições.
- **Processamento de dados financeiros:** manipulando variáveis para calcular saldos, juros e realizar projeções financeiras.
- **Desenvolvimento de jogos:** utilizando variáveis para armazenar pontos, vidas e outras informações relevantes durante a execução do jogo.

Até aqui foi possível verificar os conceitos fundamentais sobre manipulação de variáveis em Portugol. Entender como declarar, atribuir valores e manipular variáveis é essencial para desenvolver algoritmos eficientes e resolver problemas de programação.

Esses conceitos são aplicáveis em diversas áreas profissionais, desde o controle de estoque até o desenvolvimento de jogos. Ao praticar a manipulação de variáveis, você estará construindo uma base sólida necessária para se tornar um programador capacitado. Continue explorando e praticando, e você estará no caminho certo para dominar a lógica de programação.

Sequência e controle de fluxo

O entendimento da sequência e controle de fluxo é fundamental para quem está iniciando na programação de algoritmos. Em Portugol, esses conceitos são a base para criar algoritmos estruturados e eficientes. Neste sentido, exploraremos o que são as sequências e como o controle de fluxo influencia a execução de algoritmos.

Em programação, uma sequência refere-se a uma série de instruções executadas em ordem linear. Cada instrução é executada após a anterior, seguindo uma ordem predefinida. A sequência é a estrutura mais simples em programação e é usada para representar a execução e o passo a passo de tarefas.

O controle de fluxo refere-se à capacidade de direcionar o fluxo de execução do programa com base em condições específicas. Isso permite a criação de algoritmos mais flexíveis e adaptáveis, capazes de tomar decisões e repetir tarefas conforme necessário.

A seguir, podemos observar as principais características no que se refere as sequências e controle de fluxo:

- **Sequencialidade:** as instruções são executadas em ordem sequencial, uma após a outra.

ALGORITMOS E TÉCNICAS DE PROGRAMAÇÃO

- **Estruturas de decisão (condicional):** permite que o programa escolha entre diferentes caminhos de execução com base em condições lógicas. Exemplo:

se idade >= 18 então

escreva("Você é maior de idade.")

senão

escreva("Você é menor de idade.")

fimse

- **Estruturas de repetição (laços):** Permitem que um bloco de código seja repetido enquanto uma condição for verdadeira. Exemplo:

íntero contador

contador <- 1

enquanto contador <= 5 faça

escreva("Contagem: ", contador)

contador <- contador + 1

fimenquanto

Observe alguns exemplos de como as sequências e o controle de fluxo podem ser aplicados em algoritmos em diversas situações do cotidiano de clientes/empresas:

1. **Sistema de login:** utilizando estruturas condicionais para verificar se as credenciais do usuário são válidas.
2. **Análise de dados:** aplicando laços de repetição para percorrer grandes conjuntos de dados e extrair informações relevantes.
3. **Controle de estoque:** utilizando estruturas condicionais para verificar a disponibilidade de produtos e tomar decisões de reposição.

A capacidade de criar estruturas condicionais e *loops* expande as possibilidades de resolução de problemas, permitindo que os algoritmos tomem decisões e repitam tarefas de maneira inteligente.

ALGORITMOS E TÉCNICAS DE PROGRAMAÇÃO

Estes conceitos são aplicáveis em uma variedade de cenários profissionais, desde sistemas de login até análise de dados complexa. Continuemos praticando e explorando esses conceitos para solidificar sua compreensão da lógica de programação.

Siga em Frente...

Fundamentos de variáveis e constantes

Variáveis e constantes são elementos fundamentais no mundo da programação. Elas são como recipientes que armazenam valores de dados e nos permitem manipular e interagir com esses dados dentro de nossos programas, por isso o entendimento dos fundamentos desses dois elementos da programação é essencial para o futuro programador.

Sendo assim, exploraremos o que são esses fundamentos, suas características e como podem ser aplicados no desenvolvimento de algoritmos.

Variáveis são espaços na memória reservados para armazenar dados que podem ser alterados durante a execução de um programa. Cada variável possui um nome único que a identifica e um tipo que define que tipo de dado ela pode armazenar, como números inteiros, decimais, caracteres, entre outros.

Ao contrário das variáveis, as constantes são espaços na memória cujo valor não pode ser modificado durante a execução do programa. Elas são usadas para armazenar informações fixas e são declaradas com um valor constante no início do programa.

As características principais desses dois itens da programação são:

- **Nome:** tanto as variáveis quanto as constantes têm nomes que as identificam no código. Esses nomes devem seguir regras específicas, como começar com uma letra e não conter espaços.
- **Tipo de dado:** cada variável e constante é associada a um tipo de dado, indicando o formato dos dados que podem ser armazenados. Exemplos incluem inteiros, decimais, caracteres, booleanos, entre outros.
- **Atribuição:** as variáveis são inicializadas e seus valores podem ser alterados durante a execução do programa usando o operador de atribuição (`<-`). As constantes recebem um valor fixo no início e não podem ser alteradas.

Partindo da reflexão acerca de que variáveis e constantes são os pilares sobre os quais construímos nossos algoritmos, entende-se que esses dois elementos permitem que nossos programas executem cálculos, tomem decisões e interajam com o usuário.

ALGORITMOS E TÉCNICAS DE PROGRAMAÇÃO

Sem uma compreensão sólida de como utilizar variáveis e constantes, torna-se quase impossível escrever código funcional e eficaz. Dessa forma, veremos alguns exemplos de como podemos trabalhar com variáveis e constante em algoritmos e em aplicações profissionais, observe:

1. Algoritmos:

- **Manipulação de dados dinâmica:** variáveis são utilizadas para armazenar e manipular dados que podem mudar ao longo da execução do programa, como entrada do usuário ou resultados de cálculos.
- **Armazenamento de informações constantes:** constantes são úteis para armazenar informações que permanecem inalteradas durante a execução, como valores fixos utilizados em cálculos.
- **Facilidade de manutenção:** utilizar variáveis e constantes torna o código mais legível e facilita a manutenção, pois permite ajustar valores em um único local.

2. Aplicações Profissionais:

- **Cálculos financeiros:** utilizando variáveis para representar valores como saldos de contas e taxas de juros, e constantes para armazenar informações fixas, como a porcentagem de uma taxa.
- **Desenvolvimento de software:** utilizando variáveis para armazenar dados temporários durante a execução do programa e constantes para definir valores que não devem ser alterados.
- **Engenharia de software:** utilizando variáveis para representar dados dinâmicos, como entradas do usuário, e constantes para representar valores fixos usados em cálculos específicos.

Ao estudarmos sobre os fundamentos de variáveis e constantes, exploramos os pilares essenciais da programação que nos permitiram adquirir a compreensão sobre armazenar, manipular e interagir com dados de forma dinâmica e eficaz.

Compreendemos, também, a importância de utilizar variáveis e constantes de maneira adequada para tornar nossos algoritmos flexíveis, adaptáveis, legíveis e fáceis de manter. Ao dominar esses conceitos básicos fundamentais, você está preparado para avançar para tópicos mais complexos na programação. Lembre-se sempre da importância de praticar e aplicar esse conhecimento em seus projetos, pois é através da experiência prática que você se tornará um programador habilidoso e confiante.

Vamos Exercitar?

Descrição da situação-problema:

ALGORITMOS E TÉCNICAS DE PROGRAMAÇÃO

Imagine que você é responsável por desenvolver um algoritmo simples em Portugol para auxiliar no controle de estoque de uma loja de produtos eletrônicos. A loja possui diferentes tipos de produtos, cada um com sua quantidade em estoque.

A necessidade é atualizar o estoque de um produto específico com base nas entradas e saídas diárias. Crie um algoritmo que manipule variáveis para realizar esse controle de estoque de maneira eficiente.

Resolução da situação-problema:

```
1  algoritmo ControleEstoque
2
3  var
4      estoqueInicial, entradas, saidas, estoqueFinal: inteiro
5
6  inicio
7      // Informe o estoque inicial, entradas e saídas do produto
8      escreva("Informe o estoque inicial: ")
9      leia(estoqueInicial)
10
11     escreva("Informe as entradas do dia: ")
12     leia(entradas)
13
14     escreva("Informe as saídas do dia: ")
15     leia(saidas)
16
17     // Passo 2: Manipulação de Variáveis para Atualização do Estoque
18     estoqueFinal <- estoqueInicial + entradas - saidas
19
20     // Passo 3: Exibição do Estoque Atualizado
21     escreva("Estoque atualizado: ", estoqueFinal)
22
23  finalgoritmo
```

Figura 1 | Algoritmo controle de estoque.

Nesta situação-problema, você aprendeu a manipular variáveis em Portugol para criar um algoritmo de controle de estoque. O algoritmo permite a atualização do estoque com base nas entradas e saídas diárias do produto. Os passos incluíram a declaração de variáveis, a leitura de dados do usuário, a manipulação das variáveis para calcular o estoque final e, por fim, a exibição do resultado.

ALGORITMOS E TÉCNICAS DE PROGRAMAÇÃO

A prática dessa atividade proporciona uma compreensão sólida sobre como variáveis podem ser utilizadas para resolver problemas práticos no contexto de programação. Continue praticando e explorando diferentes situações para aprimorar suas habilidades de manipulação de variáveis em Portugol.

Saiba mais

Para saber mais sobre os conceitos e Introdução aos algoritmos no que diz respeito à formação de ideias para compreensão desse conteúdo, acesse em sua Biblioteca Virtual o livro [Estudo Dirigido de Algoritmos](#), Capítulo 3 “Programação com sequência”, item 3.3 - O uso de variáveis - páginas 38 a 43.

Referências

CORMEN, T. **Algoritmos – Teoria e Prática**. 3.ed. Rio de Janeiro: LTC, 2022.

CORMEN, T. **Desmistificando Algoritmos**. Grupo GEN, 2013.

FERREIRA, A. B. H. **Novo Aurélio Século XXI**: o dicionário da língua portuguesa. 3 ed. totalmente rev. e ampl. Rio de Janeiro: Nova Fronteira, 1999.

MANZANO, J. A. N. G.; OLIVEIRA, J. F. **Estudo Dirigido de Algoritmos**. [S. I.]: Editora Saraiva, 2012.

MENÉNDEZ, A. **Simplificando Algoritmos**. Grupo GEN, 2023. E-book.

SERPA, M. S.; RODRIGUES, T. N.; ALVES, Í. C.; *et al.* **Análise de Algoritmos**. Grupo A, 2021.

Aula 5

Encerramento da Unidade

Videoaula de Encerramento

Este conteúdo é um vídeo!

Para assistir este conteúdo é necessário que você acesse o AVA pelo computador ou pelo aplicativo. Você pode baixar os vídeos direto no aplicativo

ALGORITMOS E TÉCNICAS DE PROGRAMAÇÃO



para assistir mesmo sem conexão à internet.

Dica para você

Aproveite o acesso para baixar os slides do vídeo, isso pode deixar sua aprendizagem ainda mais completa.

Olá, estudante! Nesta videoaula você verá os fundamentos essenciais que impactam diretamente a prática profissional de quem atua no vasto universo da programação.

Em um mundo cada vez mais digital, compreender os fundamentos de algoritmos é como desvendar os segredos por trás da magia da tecnologia. Assim, mergulharemos nas bases dos algoritmos, explorando como essas sequências de passos lógicos são a espinha dorsal de qualquer software, desde os mais simples até os mais complexos.

Além disso, abordaremos os conceitos básicos de linguagem de programação, a ferramenta que transforma nossos algoritmos em ações concretas. A compreensão desses conceitos é crucial para qualquer profissional que deseje desenvolver suas habilidades na área de programação, independentemente do nível de experiência.

Então, assista agora e dê mais um passo em busca do conhecimento.

Ponto de Chegada

Olá, estudante! Para desenvolver a competência desta Unidade, que é "Identificar os fundamentos e conceitos básicos em linguagens de programação", devemos, primeiramente, conhecer os conceitos fundamentais que foram abordados.

Uma das competências essenciais é a capacidade de entender e identificar as situações em que o uso de algoritmos e suas técnicas são apropriadas.

O aprendizado sobre os fundamentos dos algoritmos e das linguagens de programação proporcionam uma série de benefícios e habilidades essenciais para sua formação e atuação profissional. Algumas das vantagens são: desenvolvimento do pensamento lógico e a resolução eficiente de problemas do mundo real por meio do desenvolvimento de soluções algorítmicas.

Com o avanço do conteúdo, aprendemos como avaliar a aplicação de algoritmos em diversos contextos, desde a criação e manipulação de elementos da linguagem de programação.

O entendimento dos fundamentos dos algoritmos incentiva o desenvolvimento de um pensamento lógico e estruturado. Vimos de que forma abordar problemas de maneira analítica, decompondo tarefas complexas em passos mais simples.

Essas competências não apenas capacitam a escrever código, mas também molda uma mentalidade analítica, problemática e criativa, necessária para o sucesso em diversas áreas profissionais.

ALGORITMOS E TÉCNICAS DE PROGRAMAÇÃO

É Hora de Praticar!

Vamos colocar em prática o conhecimento adquirido nesta Unidade. Para isso, veremos um estudo de caso com o cenário descrito a seguir.

Estudo de Caso: uma pequena loja de eletrônicos enfrenta desafios no controle de estoque, resultando em dificuldades na reposição de produtos e perdas financeiras devido a itens obsoletos.

O "desafio" a ser superado é a **Gestão de Estoque**: A loja precisa otimizar a gestão de estoque para evitar falta ou excesso de produtos, mantendo um equilíbrio adequado.

A otimização do estoque resolve desafios operacionais e traz uma série de benefícios tangíveis que impactam positivamente a saúde financeira e a satisfação dos clientes. Alguns desses benefícios são citados a seguir:

- Gestão eficiente.
 - Redução de perdas financeiras.
 - Aumento da satisfação do cliente.
 - Agilidade nas reposições.
-
- A compreensão dos fundamentos dos algoritmos não apenas capacita a criação de soluções eficientes, mas também promove uma abordagem estruturada na resolução de problemas. Ao mergulhar nos algoritmos, questione-se: Como a aplicação de algoritmos pode aprimorar minha capacidade de analisar e resolver desafios cotidianos?
 - As linguagens de programação são ferramentas poderosas para dar vida a conceitos e com certeza às ideias dentro dos algoritmos. Além de aprender a sintaxe, reflita sobre como diferentes linguagens influenciam a expressão de ideias e a resolução de problemas. Pergunte a si mesmo: Como a escolha da linguagem de programação pode impactar a eficácia e a clareza das minhas soluções?

Sugestão de solução

Passo 1: Identificação de tipos de dados:

- Análise dos tipos de dados necessários para registrar informações como quantidade em estoque, preço unitário e datas de entrada e saída.

Passo 2: Manipulação de variáveis:

- Desenvolvimento de algoritmos simples para manipular variáveis, como incrementar a quantidade de produtos vendidos e atualizar o estoque após uma venda.

Passo 3: Implementação em linguagem simples:

- Escolha de uma linguagem de programação de fácil aprendizado, como **Portugol**.
- Desenvolvimento de códigos simples para calcular o saldo de produtos e verificar a necessidade de reposição.

ALGORITMOS E TÉCNICAS DE PROGRAMAÇÃO

Passo 4: Testes práticos:

- Aplicação prática dos algoritmos no controle diário de entrada e saída de produtos.
- Verificação da eficácia dos algoritmos na manutenção de um estoque equilibrado.

Conclusão

Ao implementar algoritmos simples de manipulação de variáveis em Portugol, a loja de eletrônicos conseguiu melhorar significativamente sua gestão de estoque. A simplicidade na identificação de tipos de dados e manipulação de variáveis mostrou-se eficaz na prevenção de faltas e excessos de produtos.

Assim, este estudo de caso destaca como os conceitos básicos de tipos de dados e manipulação de variáveis podem ser aplicados de forma prática e acessível, trazendo benefícios tangíveis para pequenos negócios.

Neste mapa mental, apresentamos resumidamente os principais conceitos relacionados a Fundamentos dos Algoritmos e das Linguagens de Programação, dois tópicos fundamentais em algoritmos e técnicas de programação.

Convidamos você estudante, a refletir sobre os conceitos apresentados nele.

ALGORITMOS E TÉCNICAS DE PROGRAMAÇÃO

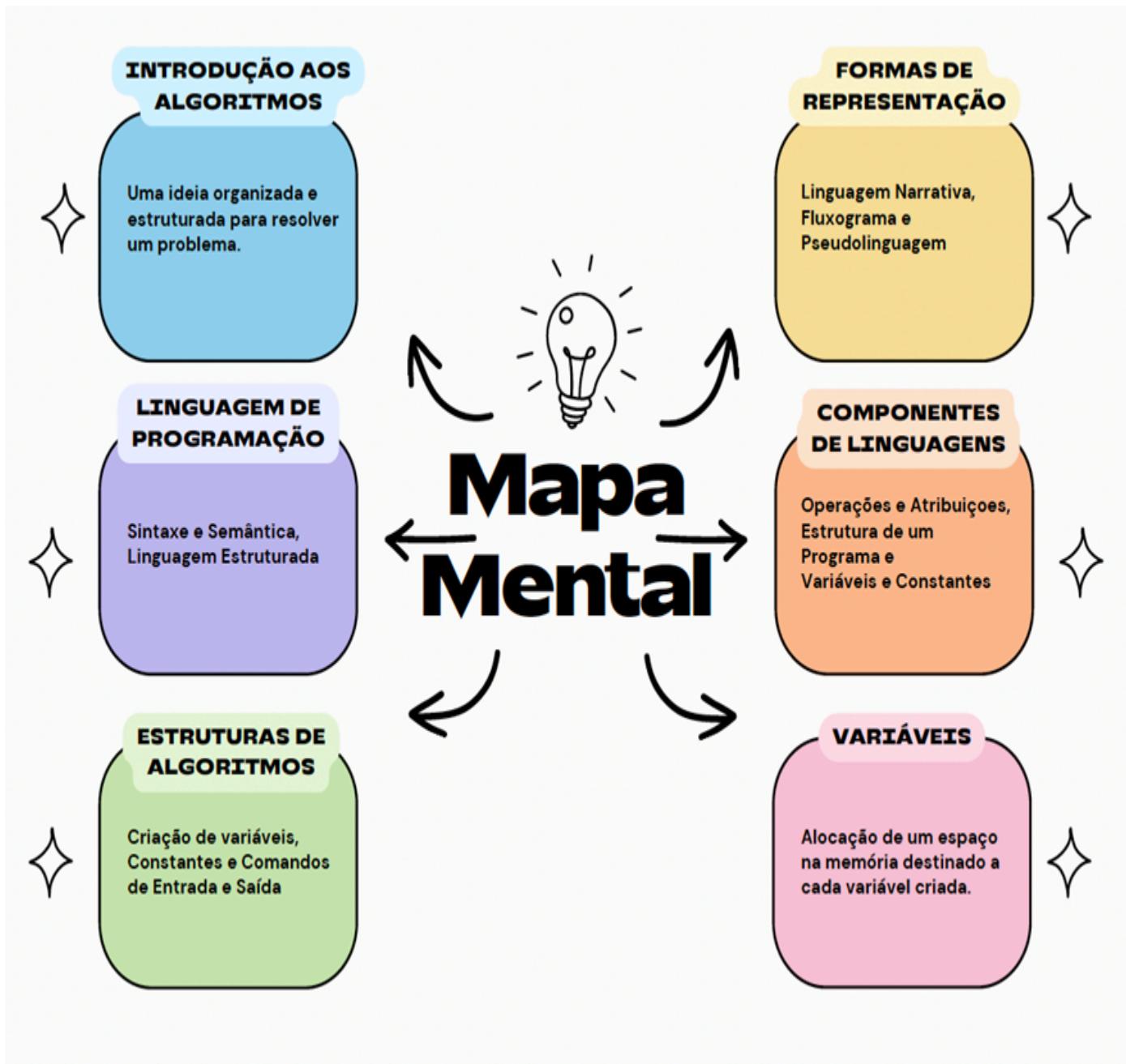


Figura | Fundamentos dos algoritmos e das linguagens de programação.

CORMEN, T. **Algoritmos – Teoria e Prática**. 3. ed. Rio de Janeiro: LTC, 2022.

CORMEN, T. **Desmistificando Algoritmos**. [S. I.]: Grupo GEN, 2013.

MANZANO, J. A. N. G.; OLIVEIRA, J. F. **Estudo Dirigido de Algoritmos**. [S. I.]: Editora Saraiva, 1997.

MENÉNDEZ, A. **Simplificando Algoritmos**. [S. I.]: Grupo GEN, 2023.

SERPA, M. S.; et al. **Análise de Algoritmos**. [S. I.]: Grupo A, 2021.

ALGORITMOS E TÉCNICAS DE PROGRAMAÇÃO

Unidade 2

Aplicação de Constantes, Variáveis, Operações e Escopo de Programação

Aula 1

Constantes e Variáveis com Tipos de Dados Primitivos

Constantes e variáveis com tipos de dados primitivos

Este conteúdo é um vídeo!



Para assistir este conteúdo é necessário que você acesse o AVA pelo computador ou pelo aplicativo. Você pode baixar os vídeos direto no aplicativo para assistir mesmo sem conexão à internet.

Dica para você

Aproveite o acesso para baixar os slides do vídeo, isso pode deixar sua aprendizagem ainda mais completa.

Olá, estudante! Nesta videoaula abordaremos assuntos relativos à algoritmos como: tipos primitivos em linguagem de programação, variáveis aplicadas em linguagem de programação e como se dá o processo de utilização entre constantes e variáveis. Tudo isso aplicado em exemplos reais, tornando seu aprendizado mais significativo.

Não deixe passar a chance de aprimorar sua bagagem de conhecimentos e ampliar sua compreensão sobre algoritmos. Acompanhe-nos nessa emocionante jornada de descobertas e fortaleça suas habilidades nessa área.

Ponto de Partida

Olá, estudante! Nesta aula, exploraremos as variáveis e constantes em Linguagem de Programação. Compreenderemos como esses elementos essenciais estão presentes na construção de algoritmos e no desenvolvimento de programas. Aprender esses conceitos são a base para manipular dados e realizar operações que dão vida a programas computacionais.

Além disso, é importante destacar a relevância dos algoritmos na atualidade. Os algoritmos são conjuntos estruturados de instruções que orientam os computadores a executar tarefas

ALGORITMOS E TÉCNICAS DE PROGRAMAÇÃO

específicas.

Portanto, tratando-se de um mundo cada vez mais digital, compreender e aplicar algoritmos tornou-se um diferencial do profissional que também pode atuar em outras áreas, desde a resolução de problemas cotidianos até o desenvolvimento de tecnologias avançadas.

Diante disso, será apresentado o desenvolvimento de um algoritmo que tem o objetivo de facilitar o embarque de passageiros em um aeroporto.

Convidamos você a iniciar a construção de uma base sólida para sua jornada no universo da programação. Vamos lá?

Vamos Começar!

O objetivo desta aula é apresentar os tipos primitivos de dados na Linguagem de Programação, variáveis aplicadas e o uso de constantes e variáveis em Linguagem de Programação, além das principais características envolvidas dentro do cenário de uma Linguagem de Programação propriamente dita.

Dentre as muitas linguagens de programação que temos no universo da programação, uma das mais conceituadas e difundidas tanto no mercado de desenvolvimento de software como na área acadêmica se encontra a Linguagem de Programação C.

A linguagem de programação C é uma das mais influentes e amplamente utilizadas na história da computação. Desenvolvida por Dennis Ritchie em meados dos anos 70, a linguagem C foi concebida para superar as limitações de linguagens anteriores e proporcionar eficiência, controle e portabilidade.

- **Definições e características:** a essência da linguagem C reside em sua simplicidade e poder. Projetada para ser de fácil aprendizado, ela oferece um conjunto robusto de recursos, incluindo ponteiros, estruturas, e uma abordagem procedural. Sua eficiência e capacidade de lidar diretamente com a memória do computador tornam-na uma escolha popular para desenvolvedores que buscam controle preciso sobre o hardware.
- **Histórico:** o ano de 1978 marcou a história da C com a publicação do livro *The C Programming Language*, escrito por Brian Kernighan e Dennis Ritchie. Este livro tornou-se uma referência padrão, contribuindo significativamente para a disseminação da linguagem. A partir desse momento, a popularidade da C cresceu exponencialmente, influenciando a criação de outras linguagens e sistemas operacionais, como o Unix.
- **Exemplo clássico:** um exemplo clássico que ilustra a simplicidade e poder da C é o programa "*Hello, World!*". Vejamos um código básico na Figura 1:

ALGORITMOS E TÉCNICAS DE PROGRAMAÇÃO

```
1 #include <stdio.h>
2
3 √ int main() {
4
5     printf("Hello, World!\n");
6     return 0;
7
8
9 }
```

Figura 1 | Hello World.

Conforme observa-se na Figura 1, a linguagem de programação C é construída com uma sintaxe simples e de fácil entendimento pelo desenvolvedor.

Neste exemplo, *printf()* é uma função que exibe a mensagem na tela, e *main()* é a função principal do programa. Este código curto, porém, significativo, destaca a clareza e a estrutura da linguagem C.

Ao longo da aula vamos explorar na prática comandos e formas da escrita da linguagem de programação C, no entanto, vamos falar um pouco dos tipos primitivos de dados na Linguagem de Programação.

Tipos primitivos de dados na Linguagem de Programação

Os tipos primitivos em linguagem de programação constituem a base fundamental para o desenvolvimento de algoritmos e sistemas. No contexto da linguagem C, esses tipos desempenham um papel importante na manipulação de dados, permitindo que programadores representem e processem informações de maneira eficaz.

Os quatro tipos primitivos no Portugol são: inteiro, real, caractere e lógico. Já na Linguagem de Programação C os tipos primitivos são: int, float ou double, char e bool. Os tipos representam a

ALGORITMOS E TÉCNICAS DE PROGRAMAÇÃO

mesma classificação, porém os nomes mudam do português, português estruturado, para o inglês, como na maioria das linguagens de programação existentes (PINHEIRO, 2012).

Tipo de Dado	Descrição	Bytes	Faixa de Valor
int	Número inteiro	4	-2,147,483,648 a 2,147,483,647
float	Número de ponto flutuante simples	4	$\pm 1.2\text{e-}38$ a $\pm 3.4\text{e}38$
double	Número de ponto flutuante duplo	8	$\pm 2.3\text{e-}308$ a $\pm 1.7\text{e}308$
char	Caractere	1	-128 a 127 (ou 0 a 255 para char sem sinal)
bool	Booleano (verdadeiro/falso)	1	true (verdadeiro) ou false (falso)

Quadro 1 | Tipos primitivos na Linguagem C.

O Quadro 1 apresenta as colunas, tipos de dados, descrição, *bytes* e faixa de valor. O tipo de dado é o nome dado a cada um, assim como a descrição, que traz o que o tipo de dado representa.

Os bytes estão relacionados à quantidade de memória esse tipo de dado ocupará desde o momento em que for criado no algoritmo. Por último, a faixa de valor representa limites de valores possíveis ou especificamente quais valores poderão ser atribuídos a esses tipos de dados.

Veremos, agora, cada tipo primitivo na linguagem C e um exemplo de criação e atribuição de variáveis no mesmo tipo de dados (PINHEIRO, 2012).

- **Inteiro(int):** o tipo inteiro é amplamente utilizado para representar números inteiros, positivos ou negativos. Em situações reais, podemos aplicá-lo no desenvolvimento de programas para contadores ou para armazenar a idade de uma pessoa. Exemplo:

```
int idade = 32;
```

ALGORITMOS E TÉCNICAS DE PROGRAMAÇÃO

- **Ponto flutuante (*float* e *double*):** tipos de ponto flutuante são essenciais para lidar com números decimais. Em aplicações práticas, podem representar valores como preços de produtos ou coordenadas geográficas. Exemplo:

```
float preco = 15.99;
```

- **Caractere (*char*):** o tipo caractere é utilizado para representar caracteres individuais. Em cenários reais, poderíamos usá-lo para armazenar a inicial de um nome. Exemplo:

```
char inicial = 'INOVAÇÃO';
```

- **Lógico (*bool*):** introduzido na linguagem C a partir da versão C99, o tipo lógico (*bool*) é vital para expressar valores verdadeiros ou falsos. Em situações práticas, pode ser aplicado em tomadas de decisão, como verificar se um usuário está logado. Exemplo:

```
bool usuarioLogado = true;
```

A compreensão dos tipos primitivos em C estabelece a base necessária para a manipulação eficiente de dados. Exploramos exemplos práticos que ilustram como esses tipos são aplicados em situações reais. No entanto, é importante destacar que os tipos primitivos são apenas o ponto de partida em um vasto mundo de programação.

Portanto, ao compreender profundamente os tipos primitivos, você estará mais preparado para enfrentar desafios mais complexos na construção de algoritmos e sistemas robustos.

Variáveis aplicadas na Linguagem de Programação

A programação de computadores é uma arte complexa que envolve a criação de algoritmos e a manipulação de dados para solucionar problemas simples ou complexos, pois ao contrário do que se pensa, existem algoritmos que são extremamente simples.

Nesse cenário, as variáveis emergem como elementos fundamentais, desempenhando o seu papel com uma lógica eficiente de programas. Ao estudarmos as variáveis na linguagem de programação C, somos conduzidos por um caminho que vai além da simples alocação de memória; é um percurso que revela o poder de dar nomes e significados aos dados, transformando-os em ferramentas dinâmicas para expressar conceitos abstratos e solucionar desafios práticos.

As variáveis, nesse contexto, são como os blocos fundamentais que compõem a estrutura de um edifício complexo. Elas oferecem um método estruturado e organizado para armazenar e acessar

ALGORITMOS E TÉCNICAS DE PROGRAMAÇÃO

informações variadas, desempenhando um papel vital na manipulação e compreensão de dados.

À medida que adentramos esse domínio, é fundamental compreendermos não apenas a sintaxe técnica das variáveis em C, mas também a filosofia subjacente que as transforma em elementos poderosos de construção de software.

Na linguagem C, a abstração de variáveis não se resume apenas à alocação de espaços de memória. Cada variável é um contêiner semântico, capaz de representar conceitos específicos em um programa.

Essa característica torna as variáveis como se fossem objetos que encapsulam significados e funcionalidades. Nesta explicação, iremos além dos aspectos técnicos, adentrando o terreno da aplicação prática e do entendimento profundo das variáveis.

Diante desse contexto, concluímos que as variáveis em C são espaços de armazenamento nomeados que representam valores de dados. Elas são fundamentais para lidar com diferentes tipos de informações em um programa. Podemos pensar nelas como repositórios de memória, onde podemos armazenar e recuperar dados conforme necessário (PINHEIRO, 2012).

- **Um exemplo prático:** considere um programa simples que calcula a média de notas de alunos. Podemos utilizar variáveis para armazenar as notas, realizar os cálculos e exibir os resultados. Nesse contexto, teríamos variáveis como nota1, nota2, média, entre outras. Cada uma dessas variáveis desempenha um papel específico no contexto do programa.
- **Exemplo de situação real:** imagine um sistema de controle de estoque em uma empresa. As variáveis poderiam representar a quantidade de produtos em estoque, o preço unitário de cada item, o total de vendas diárias etc. Dessa forma, as variáveis oferecem uma maneira eficiente de lidar com a complexidade dos dados do mundo real.

Portanto, ficou claro que o uso de variáveis na linguagem C é essencial para qualquer programador que deseja desenvolver sistemas robustos e eficazes. As variáveis oferecem flexibilidade e adaptabilidade, permitindo que os programas lidem com uma ampla gama de informações.

Além disso, ao entender como as variáveis funcionam, os desenvolvedores podem otimizar o uso da memória do sistema, tornando seus programas mais eficientes.

Siga em Frente...

Uso de constantes e variáveis em Linguagem de Programação

No desenvolvimento de aplicações sistêmicas, o uso de constantes e variáveis é incontestável e muito presente, pois qualquer algoritmo desenvolvido precisa de variáveis na sua codificação

ALGORITMOS E TÉCNICAS DE PROGRAMAÇÃO

(PINHEIRO, 2012).

Imaginemos uma empresa que gerencia o estoque de produtos, enfrentando o desafio de lidar com taxas tributárias em constante mudança por conta da legalidade. Nesse contexto, as constantes podem ser utilizadas para representar essas taxas, oferecendo uma maneira organizada e fácil de ajustar os valores conforme as alterações fiscais.

Fazendo uma breve observação, essa prática na atualidade é trabalhada com a parametrização de sistemas. Por exemplo: se uma taxa de imposto é sempre passível de sofrer alterações, devemos desenvolver sistemas para que o próprio usuário consiga realizar essas mudanças, sem que a fonte do programa tenha que sofrer alterações.

Porém, isso só é possível com a utilização de banco de dados, que neste momento é um conteúdo muito avançado para nossa aula. Portanto, neste momento vamos pensar nas constantes dentro do algoritmo para armazenar esses valores, taxas tributárias mencionadas no texto.

Além disso, considere um sistema de controle de acesso que precisa armazenar informações sobre usuários, como senhas e permissões. Aqui, o uso de variáveis torna-se essencial para garantir a flexibilidade na gestão desses dados, permitindo modificações conforme as necessidades da aplicação.

Também, levando em conta que em aplicações robustas isso também pode e deve ser controlado com o auxílio do banco de dados, local onde se concentra todas as informações de uma empresa.

Constantes

As constantes são valores fixos que não podem ser alterados durante a execução do programa. Em um sistema de simulação meteorológica, por exemplo, podemos empregar constantes para representar valores como a velocidade da luz ou a aceleração de um objeto em queda livre.

Esses valores são fundamentais para os cálculos precisos realizados pelo programa, e ao designá-los como constantes, garantimos que permaneçam inalterados ao longo do tempo. Isso não apenas simplifica o código, tornando-o mais legível, mas também facilita possíveis atualizações, já que as constantes podem ser modificadas centralmente.

Como exemplo, uma constante como uma fórmula de uma planilha no Excel, em que é preciso fazer um controle de uma aplicação que rende 1.02 % ao mês, para saber quanto a quantia de R\$ 1.500,00 vai render em 1 ano, ou 12 meses. Para isso, usa-se a taxa de 1.02 % de rendimento como constante, pois ela não será alterada pelo menos durante 12 meses. Observa a Tabela 2:

Valor	Taxa
-------	------

ALGORITMOS E TÉCNICAS DE PROGRAMAÇÃO

\$1,500.00	\$15.30
\$1,515.30	\$15.46
\$1,530.76	\$15.61
\$1,546.37	\$15.77
\$1,562.14	\$15.93
\$1,578.08	\$16.10
\$1,594.17	\$16.26
\$1,610.43	\$16.43
\$1,626.86	\$16.59
\$1,643.45	\$16.76
\$1,660.22	\$16.93
\$1,677.15	\$17.11

Tabela 2 | Taxa fixa.

A coluna Taxa vai mudando conforme os valores, porém o percentual permanece fixo em 1.02%, pois 15,30 é exatamente 1.02% em relação a 1500, assim como, 17,46 é exatamente 1.02%, porém agora em relação a 1515.30. Dessa maneira utiliza-se o valor do percentual como constante.

Se criarmos um algoritmo na linguagem C que represente esse exemplo, sabemos que é possível utilizar o valor do percentual da taxa como constante.

Variáveis

Por outro lado, as variáveis são elementos dinâmicos que armazenam dados modificáveis durante a execução do programa. Elas são mutáveis, passível de alteração em seu valor e muito utilizadas no desenvolvimento de softwares complexos e em algoritmos que são utilizados em programas de baixa complexidade.

Outro fator na utilização de variáveis é de que é praticamente impossível a criação de qualquer programa sem a sua utilização.

No desenvolvimento de software, o uso eficiente de variáveis reflete diretamente na qualidade e na adaptabilidade do sistema. A sua padronização simplifica a manutenção do código, uma vez que modificações em seus valores são concretizadas durante a execução do programa. Isso reduz o risco de erros e agiliza o processo de desenvolvimento.

ALGORITMOS E TÉCNICAS DE PROGRAMAÇÃO

Portanto, o uso inteligente de variáveis permite a criação de sistemas mais dinâmicos, capazes de lidar com uma ampla gama de dados e situações.

Basicamente as variáveis durante o desenvolvimento em qualquer linguagem de programação se dá juntamente com os tipos de dados, isto porque, no momento de criação da variável em uma linguagem "fortemente TIPADA", que é o caso da linguagem C, temos que classificá-la quanto ao tipo de informação que ela pode trabalhar.

Então, criamos uma variável qualquer e a definimos como int (inteira), portanto, estamos definindo que as informações trafegadas no interior dessa variável obedecerão ao tipo definido, que são neste caso: valores numéricos inteiros, podendo incluir valores negativos e positivos incluindo o numeral 0 (zero) (PINHEIRO, 2012).

Para exemplificar melhor, vamos aos exemplos por meio da linguagem de programação C. Primeiramente com o exemplo de uma variável do tipo inteiro, Figura 2:

```
1 #include <stdio.h>
2
3 ∵ int main() {
4     // Declaração de uma variável inteira
5     int idade;
6
7     // Atribuição de um valor à variável
8     idade = 25;
9
10    // Impressão do valor da variável
11    printf("Idade: %d\n", idade);
12
13    return 0;
14
15 }
```

Figura 2 | Tipo int.

ALGORITMOS E TÉCNICAS DE PROGRAMAÇÃO

No exemplo, a variável int do tipo inteiro é criada. Na linha 28 ele recebe um novo valor com o operador de atribuição e na linha 11 ela é apresentada ao usuário.

Na Figura 3, temos um exemplo de criação e manipulação de variável do tipo fracionária, ou ponto flutuante.

```
1 #include <stdio.h>
2
3 ∵ int main() {
4     // Declaração de uma variável de ponto flutuante
5     float altura;
6
7     // Atribuição de um valor à variável
8     altura = 1.75;
9
10    // Impressão do valor da variável
11    printf("Altura: %.2f metros\n", altura);
12
13    return 0;
14 }
```

Figura 3 | Tipo float.

No exemplo, a variável altura do tipo *float* é criada e na linha 8 recebe um novo valor (1.75) com o operador de atribuição e na linha 11 ela é apresentada ao usuário em conjunto com uma mensagem de informação na tela.

Na Figura 4, observe um exemplo de criação e manipulação de variável do tipo caractere.

ALGORITMOS E TÉCNICAS DE PROGRAMAÇÃO

```
1 #include <stdio.h>
2
3 ∵ int main() {
4
5     // Declaração de uma variável de caractere
6     char letra;
7
8     // Atribuição de um valor à variável
9     letra = 'A';
10
11    // Impressão do valor da variável
12    printf("Letra: %c\n", letra);
13
14    return 0;
15
16 }
17
```

Figura 4 | Tipo char.

No exemplo, a variável letra do tipo *char* é criada e na linha 6 recebe um novo valor, a letra 'A' em maiúscula. Observe que a atribuição é feita com a utilização dos caracteres de aspas simples (''). Finalizando o exemplo na linha 12, o valor da variável letra é apresentado ao usuário na tela do programa.

Para encerrar os exemplos com os tipos de dados primitivos na linguagem de programação C, temos o tipo boolean, conhecido no Portugol como tipo de dados "Lógicos". Observe a Figura 5.

ALGORITMOS E TÉCNICAS DE PROGRAMAÇÃO

```
1 #include <stdio.h>
2 #include <stdbool.h>
3
4 ∵ int main() {
5     // Declaração de uma variável lógica
6     bool estaChovendo;
7
8     // Atribuição de um valor à variável
9     estaChovendo = false;
10
11    // Impressão do valor da variável
12    printf("Está chovendo? %s\n", estaChovendo ? "Sim" : "Não");
13
14    return 0;
15
16 }
```

Figura 5 – Tipo boolean.

No exemplo, a variável altura do tipo boolean é criada e na linha 9 recebe um novo valor. Relembrando, variáveis do tipo boolean só podem armazenar dois valores e ainda um de cada vez.

Os valores são "true" ou "false", que em português correspondem a VERDADEIRO ou FALSO. Portanto, na linha 9 recebe o valor "false" e na sequência, linha 12, o conteúdo desse valor também é exibido na tela do programa.

Em síntese, o entendimento adequado do uso de constantes e variáveis na linguagem de programação, especialmente na linguagem C, é necessário para o desenvolvimento eficiente de sistemas.

A capacidade de aplicar esses conceitos de maneira estratégica não apenas simplifica o processo de programação, mas também contribui para a criação de soluções mais robustas e adaptáveis. Ao valorizar a organização do código por meio de constantes e a flexibilidade proporcionada por variáveis, os desenvolvedores estão capacitados a enfrentar os desafios complexos do desenvolvimento de software, garantindo sistemas mais sólidos e eficazes.

Vamos Exercitar?

Descrição da situação-problema:

ALGORITMOS E TÉCNICAS DE PROGRAMAÇÃO

Imagine que você trabalha no desenvolvimento de um sistema para facilitar o processo de embarque em um aeroporto. Nesse contexto, é essencial lidar com diferentes tipos primitivos em Linguagem de Programação para armazenar informações específicas de cada passageiro, como nome, número do bilhete, idade e necessidade de assistência especial.

Resolução da situação-problema:

Identificação dos Dados:

- Enumere os tipos primitivos necessários, como caractere para o nome, inteiro para o número do bilhete e a idade, e lógico para indicar se o passageiro necessita de assistência especial.

Entrada de Dados:

- Descreva o processo de coleta de informações, destacando a importância de solicitar o nome, número do bilhete, idade e se há necessidade de assistência especial.

Processamento:

- Não codifique, mas mencione que o sistema deve realizar verificações, como a idade mínima permitida para o embarque, garantindo que apenas passageiros elegíveis embarquem.

Exibição de Resultados:

- Indique que o sistema deve fornecer feedback ao passageiro, informando se o embarque é permitido ou se há alguma restrição devida à idade ou necessidade de assistência especial.

Com o desenvolvimento do processo de embarque de passageiros, aplicando tipos de dados primitivos dentro da Linguagem de Programação, é fundamental para garantir a segurança e a organização nos aeroportos.

Essa situação-problema destaca como a compreensão e o uso adequado dos tipos primitivos são essenciais para a construção de sistemas que lidam com informações variadas de forma coerente, otimizando a experiência de embarque para passageiros e equipes.

Saiba mais

Para saber mais sobre os conceitos e formas de aplicação de tipos de dados na linguagem de programação C e ainda, quais suas variações e funcionalidades, acesse em sua Biblioteca Virtual

ALGORITMOS E TÉCNICAS DE PROGRAMAÇÃO

o livro [*Elementos de Programação em C*](#), Capítulo 3, 4 e 5 - páginas 52 a 129.

Referências

CORMEN, T. **Algoritmos – Teoria e Prática**. 3.ed. Rio de Janeiro: LTC, 2022.

CORMEN, T. **Desmistificando Algoritmos**. Grupo GEN, 2013.

FERREIRA, A. B. H. **Novo Aurélio Século XXI: o dicionário da língua portuguesa**. 3 ed. totalmente rev. e ampl. Rio de Janeiro: Nova Fronteira, 1999.

MANZANO, J. A. N G.; OLIVEIRA, J. F. **Estudo Dirigido de Algoritmos**. 15. ed. Editora Saraiva, 2012.

MENÉNDEZ, A. **Simplificando Algoritmos**. Grupo GEN, 2023.

PINHEIRO, F. A. C. **Elementos de Programação em C**. Porto Alegre: Bookman, 2012.

SERPA, M. S.; RODRIGUES, T. N.; ALVES, I. C.; *et al.* **Análise de Algoritmos**. Grupo A, 2021

Aula 2

Variáveis com Tipos de Dados Composto e Ponteiros

Variáveis com tipos de dados compostos e ponteiros

Este conteúdo é um vídeo!



Para assistir este conteúdo é necessário que você acesse o AVA pelo computador ou pelo aplicativo. Você pode baixar os vídeos direto no aplicativo para assistir mesmo sem conexão à internet.

Dica para você

Aproveite o acesso para baixar os slides do vídeo, isso pode deixar sua aprendizagem ainda mais completa.

Olá, estudante!

ALGORITMOS E TÉCNICAS DE PROGRAMAÇÃO

É com entusiasmo que convido você a participar de mais uma videoaula dessa unidade de ensino. Lembrando que a aula está alinhada com as competências e habilidades a serem desenvolvidas. É necessário continuar em constante evolução para o aprendizado de novos elementos e técnicas para no desenvolvimento de algoritmos.

Além disso, abordaremos assuntos relativos à algoritmos como: variáveis compostas e endereço de memória, variáveis dos tipos: string, vetor e matriz, uso de variáveis do tipo ponteiro.

Ponto de Partida

Olá, estudante! Nesta aula, iniciaremos nossa descoberta em outros tipos de variáveis presentes na em Linguagem de Programação. O objetivo é compreender esses elementos e como eles auxiliam na construção de algoritmos e programas de computadores. Aprender esses conceitos serão fundamentais para trabalhar com tipos de dados mais complexos em linguagem de programação.

Um dos conteúdos-chave desta aula é o trabalho com "ponteiros". Eles contêm características únicas e poderosas que por conta dessas qualidades, oferecem aos programadores um nível mais profundo de controle sobre a manipulação de dados e a gestão de memória.

Para contextualizar, será apresentado o desenvolvimento de um exemplo envolvendo ponteiros.

Convidamos você a iniciar os conteúdos que são responsáveis para a contribuição da sua formação técnica em desenvolvimento de sistemas. Vamos lá?

Vamos Começar!

O objetivo desta aula é apresentar as variáveis compostas e endereço na memória, assim como, variáveis do tipo *string*, vetores e matrizes e o uso de variáveis do tipo ponteiro na Linguagem de Programação C.

Para iniciar essa aula, vamos começaremos falando sobre variáveis compostas. Em Aulas anteriores, falamos sobre variáveis e constantes, definimos os tipos de dados primitivos na linguagem C e alguns exemplos sobre criação, inicialização e manipulação de valores.

A partir de agora, vamos aprofundar um pouco mais este assunto abordando variáveis compostas e endereço de memória.

Variáveis compostas e endereço na memória

Variáveis são elementos fundamentais em programação, armazenando dados que podem ser manipulados durante a execução de um programa. Em linguagem de programação C, as variáveis

ALGORITMOS E TÉCNICAS DE PROGRAMAÇÃO

compostas são estruturas que podem armazenar diferentes tipos de dados, oferecendo maior flexibilidade na organização e manipulação de informações.

Sobre as variáveis compostas, podemos elencar:

- **Structs (Estruturas):** em linguagem de programação C, uma variável composta é uma estrutura de dados que permite agrupar diferentes tipos de informações sob um único nome. Entre essas variáveis compostas, destaca-se a *struct* (estrutura), que desempenha um papel essencial na organização e manipulação de dados. A *struct* possibilita a criação de tipos de dados personalizados, incorporando diversos elementos com tipos distintos em uma única entidade. Entre suas características destaca-se:
 - **Agrupamento Lógico**, agrupando dados logicamente relacionados como por exemplo: ao modelar um aluno, podemos agrupar seu nome, idade e média de notas em uma única *struct* Aluno.
 - **Acesso Simples**, os membros da *struct* são acessados de forma simples, utilizando o operador ponto (.). Isso facilita a manipulação de dados, permitindo a referência direta a cada elemento.
 - **Customização de Tipos**, a *struct* possibilita a criação de tipos de dados personalizados. Com ela, podemos definir estruturas que melhor se adequam aos requisitos específicos de um programa. A Figura 1 apresenta um exemplo de uma *struct* na linguagem C.

```

3 < struct Aluno {
4   char nome[50];
5   int idade;
6   float mediaNotas;
7 };
  
```

Figura 1 | Structs (Estrutura).

- **Arrays (Vetores):** Em linguagem de programação C, um *array* (ou vetor) é uma variável composta que armazena elementos do mesmo tipo em uma estrutura sequencial. Essa estrutura é caracterizada por um nome único e um índice numérico que permite o acesso e

ALGORITMOS E TÉCNICAS DE PROGRAMAÇÃO

manipulação eficiente dos dados armazenados. Em outras palavras é uma variável que permite o armazenar várias informações do mesmo tipo em uma única variável. Exemplo: ao invés de criar três variáveis para armazenar o nome de 3 pessoas, com o vetor é possível criar somente uma variável. E nessa variável do tipo vetor, armazenar os 3 nomes separando-os por índices. Podemos citar algumas características do *array*:

- **Armazenamento Sequencial**, os elementos do array são armazenados de forma sequencial na memória, ocupando posições contíguas. Isso permite acesso direto a cada elemento por meio de seu índice.
- **Tamanho Fixo**, o tamanho do *array* é fixo durante a execução do programa e precisa ser definido no momento de sua declaração como variável.
- **Acesso por índice**, os elementos de um *array* são acessados por meio de índices inteiros. O primeiro elemento possui índice 0, o segundo índice 1 e assim por diante. Esse acesso facilita a manipulação e processamento dos dados.

A Figura 2 apresenta um exemplo de um *array* na linguagem C. Observe que é criado o array *numeros* com 5 elementos, após isso são atribuídos os 5 primeiros números ao array em ordem sequencial, como pode ser observado na linha 2 da Figura2.

```

1
2     int numeros[5] = {1, 2, 3, 4, 5};
3
4     struct Aluno {
5         char nome[50];
6         int idade;
7         float mediaNotas;
8     };
9

```

Figura 2 | Structs (Estrutura).

- ***Unions (Uniões)***: As *unions* (uniões) em linguagem de programação C são variáveis compostas que permitem o armazenamento de diferentes tipos de dados em uma única estrutura. Ao contrário dos *structs*, em que cada membro possui seu espaço independente, em uma *union*, todos os membros compartilham o mesmo espaço de armazenamento. Entre suas características, destacam-se:

ALGORITMOS E TÉCNICAS DE PROGRAMAÇÃO

- **Espaço compartilhado**, todos os membros de uma *union* compartilham o mesmo espaço de armazenamento, o que significa que apenas um membro pode ser utilizado por vez.
- **Tamanho determinado pelo maior membro**, o tamanho total de uma *union* é determinado pelo tamanho do maior membro contido nela. Isso ajuda a garantir que a *union* seja grande o suficiente para armazenar seu maior componente.
- **Uso eficiente de memória**, as *unions* são úteis quando se deseja economizar espaço de memória. Como apenas um membro é utilizado por vez, a *union* pode ser uma alternativa eficiente para armazenar diferentes tipos de dados em situações específicas.
- **Acesso aos membros**: os membros de uma *union* são acessados diretamente, assim como em uma *struct*. A distinção entre *structs* e *unions* reside principalmente na forma como a memória é organizada. A Figura 3 apresenta um exemplo de uma *struct* na linguagem C (PINHEIRO, 2012).

```
2 < union Dado {  
3  
4     int inteiro;  
5     float decimal;  
6  
7 };  
8
```

Figura 3 | Unions (Uniões).

Em um exemplo prático utilizando a variável composta *structs*, podemos pensar na seguinte situação: imagine que precisamos representar informações sobre um estudante, incluindo nome, idade e notas. Neste caso, podemos criar uma estrutura que encapsula essas informações. Observe a Figura 4.

ALGORITMOS E TÉCNICAS DE PROGRAMAÇÃO

```

1  #include <stdio.h>
2
3  struct Aluno {
4      char nome[50];
5      int idade;
6      float mediaNotas;
7  };
8
9  int main() {
10     struct Aluno aluno1; // Declaração da variável composta do tipo struct Aluno
11
12     // Atribuição de valores aos membros da struct
13     strcpy(aluno1.nome, "João");
14     aluno1.idade = 20;
15     aluno1.mediaNotas = 8.5;
16
17     // Exibição dos dados
18     printf("Nome: %s\n", aluno1.nome);
19     printf("Idade: %d\n", aluno1.idade);
20     printf("Média de Notas: %.2f\n", aluno1.mediaNotas);
21
22     return 0;
23 }

```

Figura 4 | Informações de um Estudante.

Ao explorarmos as variáveis compostas em linguagem de programação C, notamos a versatilidade e eficiência que cada uma delas oferece em diferentes contextos. As *structs* proporcionam a capacidade de agrupar membros de tipos diferentes, criando estruturas organizadas e acessíveis.

Os *arrays* expandem essa ideia, permitindo o armazenamento sequencial de elementos do mesmo tipo, facilitando operações em conjuntos de dados.

As *unions*, por sua vez, destacam-se pela economia de memória, possibilitando o compartilhamento de espaço entre membros de diferentes tipos. Essa característica é particularmente valiosa em situações em que apenas um tipo de dado é relevante em um dado momento, proporcionando flexibilidade e otimização de recursos.

Unindo essas variáveis compostas, torna-se possível criar estruturas de dados complexas e adaptáveis. Por exemplo, podemos ter uma *struct* que contém um *array* de elementos e, dentro dela, uma *union* para lidar com diferentes tipos de dados. Essa combinação permite a criação de estruturas hierárquicas que representam eficientemente situações do mundo real.

ALGORITMOS E TÉCNICAS DE PROGRAMAÇÃO

Resumindo, ao compreendermos as características e aplicações específicas de *structs*, *arrays* e *unions*, capacitamo-nos para tomar decisões informadas ao projetar e implementar estruturas de dados em nossos programas.

Siga em Frente...

Variáveis dos tipos: *string*, vetor e matriz

Na vasta paisagem da programação, a linguagem C se destaca por sua eficiência e capacidade de manipular dados de forma direta. Um dos pilares fundamentais dessa linguagem são as variáveis, que não se limitam aos tipos primitivos. Neste contexto, exploraremos estruturas de dados mais avançadas, como *strings*, vetores e matrizes, que expandem significativamente as possibilidades de armazenamento e manipulação de informações.

Começaremos compondo as *strings* em C. Essas sequências de caracteres são essenciais para lidar com texto, apresentando-se como uma ferramenta versátil para representar palavras, frases e até mesmo documentos inteiros. A peculiaridade das *strings* em C reside no tratamento como *arrays* de caracteres, revelando uma abordagem única e eficaz para a manipulação de texto.

Na sequência, os vetores, ou *arrays* unidimensionais, que desempenham um papel crucial na organização de dados homogêneos. A flexibilidade oferecida pelos vetores permite a criação de estruturas capazes de armazenar múltiplos elementos do mesmo tipo, proporcionando um ambiente propício para lidar com coleções de informações de maneira estruturada e eficiente.

À medida que nos aprofundamos, encontramos as matrizes, uma extensão natural dos vetores. Com a capacidade de representar *arrays* bidimensionais, as matrizes organizam os dados em linhas e colunas, proporcionando uma abordagem sistemática para manipular conjuntos de informações mais complexos. Os dois índices necessários para acessar elementos em matrizes oferecem uma perspectiva mais rica na representação de dados multidimensionais.

Vamos explorar essas estruturas de dados para entender como elas podem ser aplicadas em diversas situações.

***Strings* em C**

As *strings* são sequências de caracteres, e em C, são tratadas como *arrays* de caracteres, terminadas por um caractere nulo \0. Essa estrutura é fundamental para lidar com texto, desde simples saudações até manipulação de grandes blocos de informações.

A biblioteca padrão de C fornece funções específicas para operações em *strings*, como **strcpy**, **strcat** e **strlen**, facilitando a manipulação desses dados. Um exemplo de codificação com *strings*

ALGORITMOS E TÉCNICAS DE PROGRAMAÇÃO

em C pode ser observado na Figura 5 (PINHEIRO, 2012).

```
1  ✓ #include <stdio.h>
2    #include <string.h>
3
4  ✓ int main() {
5
6      char saudacao[] = "Olá";
7      char nome[20];
8
9      printf("Digite seu nome: ");
10     scanf("%s", nome);
11
12     strcat(saudacao, nome);
13     printf("%s\n", saudacao);
14
15     return 0;
16
17 }
```

Figura 5 | Strings em C.

Vetores em C

Os vetores, ou *arrays* unidimensionais, permitem armazenar múltiplos elementos do mesmo tipo, acessíveis por meio de índices. São ideais para situações em que precisamos lidar com coleções de dados homogêneas.

ALGORITMOS E TÉCNICAS DE PROGRAMAÇÃO

A declaração de um vetor em C envolve especificar o tipo dos elementos e o tamanho do *array* (PINHEIRO, 2012). Um exemplo de codificação com Vetores em C pode ser observado na Figura 6.

```
1 #include <stdio.h>
2
3 int main() {
4
5     int numeros[5] = {1, 2, 3, 4, 5};
6
7     for (int i = 0; i < 5; i++) {
8         printf("%d ", numeros[i]);
9     }
10
11    return 0;
12
13 }
```

Figura 6 | Vetores em C.

Matrizes em C

As matrizes são extensões dos vetores, representando *arrays* bidimensionais, elas permitem organizar dados em linhas e colunas, sendo declaradas especificando o tipo dos elementos, o número de linhas e o número de colunas.

O acesso a elementos em matrizes requer dois índices, proporcionando uma abordagem estruturada para lidar com conjuntos de dados bidimensionais (PINHEIRO, 2012). Um exemplo de codificação com Matrizes em C pode ser observado na Figura 7.

ALGORITMOS E TÉCNICAS DE PROGRAMAÇÃO

```
1 #include <stdio.h>
2
3 int main() {
4
5     int matriz[2][3] = {{1, 2, 3}, {4, 5, 6}};
6
7     for (int i = 0; i < 2; i++) {
8         for (int j = 0; j < 3; j++) {
9             printf("%d ", matriz[i][j]);
10            }
11            printf("\n");
12        }
13
14    return 0;
15 }
```

Figura 7 | Matrizes em C.

Nesta exploração das variáveis mais avançadas em C, visamos não apenas compreender os conceitos fundamentais, mas também demonstrar a aplicação prática dessas estruturas.

Ao final dessa jornada, os programadores terão uma compreensão sólida das variáveis mais complexas em C, fortalecendo suas habilidades para enfrentar desafios mais intrincados e construir soluções robustas em seus programas. Através de exemplos práticos e contextuais, buscamos fornecer uma visão abrangente e aprofundada dessas poderosas ferramentas.

Compreender e dominar o uso de *strings*, vetores e matrizes em C é essencial para desenvolver programas eficientes e capazes de lidar com dados de maneira flexível. As *strings* oferecem uma abordagem prática para trabalhar com texto, enquanto vetores e matrizes fornecem estruturas sólidas para manipulação de coleções de dados.

O programador C, ao explorar essas estruturas, adquire habilidades fundamentais para enfrentar desafios diversos na programação.

Uso de variáveis do tipo ponteiro

ALGORITMOS E TÉCNICAS DE PROGRAMAÇÃO

Na linguagem de programação C, as variáveis do tipo ponteiro são elementos fundamentais para o controle preciso da memória e manipulação eficiente de dados. Diferentemente de outras linguagens, C oferece aos programadores a capacidade de interagir diretamente com os endereços de memória, proporcionando uma outra prática no desenvolvimento de algoritmos.

Neste contexto, exploraremos o conceito de variáveis ponteiro, examinando como elas contribuem para a dinâmica e os recursos dos programas escritos na linguagem de programação C.

Definição de ponteiros: em C, um ponteiro é uma variável cujo valor é o endereço de outra variável. Em vez de armazenar diretamente dados, um ponteiro armazena a localização na memória onde esses dados estão.

Essa capacidade de apontar para áreas específicas da memória oferece controle direto e preciso sobre os dados. É importante entender que é exatamente dessa forma que nos referimos aos ponteiros, como um "APONTADOR", pois realmente é como eles se comportam (PINHEIRO, 2012).

A declaração de ponteiros em C segue a sintaxe tipo `*nome_do_ponteiro`. Por exemplo, para declarar um ponteiro para um inteiro, utilizamos `int *ptr`. Isso indica que `ptr` é um ponteiro que pode armazenar o endereço de uma variável do tipo inteiro. É importante nos atentarmos para o caractere asterisco `"*"`, ele, na declaração de variáveis, está indicando que se trata de uma variável diferente, ou seja, um ponteiro (PINHEIRO, 2012).

Uma das aplicações mais poderosas dos ponteiros em C está na alocação dinâmica de memória usando as funções `malloc`, `calloc` e `realloc`. Essas funções permitem reservar e liberar espaço de memória conforme necessário durante a execução do programa, proporcionando flexibilidade fundamental.

Ponteiro e *arrays* em C estão intrinsecamente conectados. O nome de um *array* é, na verdade, um ponteiro constante para o primeiro elemento do *array*. Isso facilita a manipulação eficiente de *arrays* e *strings*, permitindo iterações e modificações diretas nos dados.

Outro recurso avançado é a utilização de ponteiros para funções. Isso possibilita passar funções como argumentos para outras funções, proporcionando uma estratégia eficaz para implementar técnicas mais avançadas de desenvolvimento.

As variáveis do tipo ponteiro desempenham um papel fundamental na programação em C, possibilitando uma manipulação direta da memória e oferecendo um controle preciso sobre os dados. A alocação dinâmica de memória, a manipulação eficiente de *arrays* e *strings*, e a utilização de ponteiros para funções são apenas algumas das aplicações avançadas que destacam a versatilidade e potencial dessa característica.

ALGORITMOS E TÉCNICAS DE PROGRAMAÇÃO

Ao compreender profundamente o uso de ponteiros, os programadores podem desenvolver programas mais eficientes, economizando recursos e adaptando-se às necessidades dinâmicas do sistema.

No entanto, é importante manter uma abordagem cuidadosa para evitar erros relacionados à manipulação inadequada de ponteiros, como vazamento de memória. O conhecimento sólido sobre variáveis do tipo ponteiro é, portanto, essencial para programadores que buscam aprimorar suas habilidades em C e criar códigos eficientes e robustos.

Vamos Exercitar?

Descrição da situação-problema:

Imagine que você é um desenvolvedor de software em uma empresa especializada em sistemas embarcados. Sua tarefa é criar um programa que gerencie informações sobre sensores utilizados em dispositivos para IOT (Internet das Coisas). Para otimizar a gestão de dados, você decide utilizar ponteiros na linguagem "C", e assim, manipular eficientemente as informações dos sensores.

Problema: Criar um ponteiro que armazene o valor da temperatura medida por um sensor. Além disso, é necessário atribuir um valor inicial de temperatura a esse sensor.

Resolução da situação-problema:

Criação do Ponteiro:

- Inicie declarando uma variável do tipo inteiro que representará o valor da temperatura.
- Em seguida, declare um ponteiro que apontará para a variável da temperatura.

Entrada de Dados:

- Atribua um valor inicial à variável de temperatura. Por exemplo, suponha que o valor inicial seja 25 graus Celsius.
- Utilize o ponteiro para atribuir o valor da temperatura ao qual ele aponta.

Neste exemplo prático, a criação e manipulação de ponteiros foram fundamentais para o gerenciamento eficiente das informações de temperatura do sensor. O uso de ponteiros na linguagem "C" oferece flexibilidade e otimização de recursos, aspectos importantes ao lidar com sistemas embarcados.

Compreender como criar e atribuir valores a ponteiros é uma habilidade valiosa para desenvolvedores que desejam maximizar a eficiência e o controle em suas aplicações.

ALGORITMOS E TÉCNICAS DE PROGRAMAÇÃO

Saiba mais

Para aprofundar seu conhecimento sobre variáveis complexas como Ponteiros e Vetores na linguagem de programação C, e quais suas características, acesse a Biblioteca Virtual, o livro [Elementos de Programação em C](#), Capítulo 10 - páginas 229 a 273.

Referências

CORMEN, T. **Algoritmos – Teoria e Prática**. 3.ed. Rio de Janeiro: LTC, 2022.

CORMEN, T. **Desmistificando Algoritmos**. Grupo GEN, 2013.

FERREIRA, A. B. H. **Novo Aurélio Século XXI: o dicionário da língua portuguesa**. 3 ed. totalmente rev. e ampl. Rio de Janeiro: Nova Fronteira, 1999.

MANZANO, J. A. N G.; OLIVEIRA, J. F. **Estudo Dirigido de Algoritmos**. 15. ed. Editora Saraiva, 2012.

MENÉNDEZ, A. **Simplificando Algoritmos**. Grupo GEN, 2023.

PINHEIRO, F. A. C. **Elementos de Programação em C**. Porto Alegre: Bookman, 2012.

SERPA, M. S.; RODRIGUES, T. N.; ALVES, Í. C.; *et al.* **Análise de Algoritmos**. Grupo A, 2021.

Aula 3

Operações e Expressões

Operações e expressões

ALGORITMOS E TÉCNICAS DE PROGRAMAÇÃO



Este conteúdo é um vídeo!

Para assistir este conteúdo é necessário que você acesse o AVA pelo computador ou pelo aplicativo. Você pode baixar os vídeos direto no aplicativo para assistir mesmo sem conexão à internet.

Dica para você

Aproveite o acesso para baixar os slides do vídeo, isso pode deixar sua aprendizagem ainda mais completa.

Olá, estudante! Nesta videoaula abordaremos assuntos relativos a operações matemáticas e booleanas, a aplicação de operações e expressões em linguagem C e outros assuntos relacionados.

Não deixe passar a chance de assistir essa aula e de aprimorar seus conhecimentos e sua compreensão sobre algoritmos. Vamos juntos, nesse caminho de descobertas emocionantes, e assim, fortalecendo suas habilidades nessa área.

Ponto de Partida

Olá, estudante! Nesta aula, estudaremos conteúdos como expressões matemáticas e booleanas e funções predefinidas para linguagem de programação. Prepare-se para compreender como esses elementos essenciais desempenham um papel fundamental na construção de algoritmos e no desenvolvimento de programas.

Aprender esses conceitos são a base para trabalhar com cálculos dentro dos algoritmos realizando operações e procedimento, peças-chave para a lógica de sistemas computacionais.

Além disso, é importante destacar a relevância dos algoritmos na atualidade. Vale lembrar que algoritmo é um conjunto de passos ordenados para resolução de um problema. Eles também são utilizados na construção de diferentes caminhos para que a solução seja apresentada.

Diante disso, será apresentado o desenvolvimento de um algoritmo que tem o objetivo de realizar cálculos matemáticos por meio de operações matemáticas trazendo a solução para o problema proposto.

Vamos lá?

Vamos Começar!

ALGORITMOS E TÉCNICAS DE PROGRAMAÇÃO

O objetivo desta Aula é apresentar as operações matemáticas booleanas, as operações em linguagem de programação C e as funções predefinidas na linguagem, além das principais características envolvidas neste conteúdo em relação ao cenário de desenvolvimento de softwares.

A história das operações matemáticas booleanas remonta ao século XIX, quando o matemático britânico George Boole desenvolveu a lógica booleana, uma teoria matemática que trata da manipulação de valores lógicos e da relação entre eles. Boole introduziu conceitos fundamentais, como as operações AND, OR e NOT, que mais tarde se tornaram pilares essenciais na área da computação.

Essas operações são conhecidas como operações lógicas e são muito utilizadas com outras estruturas, como por exemplo a estruturas de seleção. Em uma estrutura de seleção, é muito comum ser necessário mais do que uma condição para validar a execução de comandos. Imagine que um pai diga para seu filho, que se ele passar de ano na faculdade e se não tiver até 3 faltas em todo o semestre, ele vai ganhar um carro. Dessa maneira, para que ele ganhe o carro é preciso passar pelas 2 condições, pois ele se ele passar de ano na faculdade, mas tiver 5 faltas ele não vai ganhar o carro. É para essas situações que os operadores lógicos funcionam dentro dos algoritmos.

Lógica Booleana e a Realidade Profissional

A lógica booleana é a base do sistema digital que impulsiona a tecnologia moderna. Em cenários profissionais, essas operações são amplamente aplicadas na programação, na eletrônica e em sistemas de controle. Por exemplo, em sistemas de automação industrial, as operações booleanas são utilizadas para criar lógicas de controle que garantem o funcionamento eficiente de máquinas e processos.

As operações matemáticas booleanas incluem:

- **&& (E)**: Retorna verdadeiro se ambas as condições forem verdadeiras.
- **|| (OU)**: Retorna verdadeiro se pelo menos uma condição for verdadeira.
- **NOT (NÃO)**: Inverte o valor lógico, de verdadeiro para falso e vice-versa.

A lógica booleana é fundamental para expressar condições e tomar decisões em algoritmos. Ela permite criar estruturas de controle de fluxo, como if, else e switch, para direcionar o comportamento do programa com base em condições específicas.

Vamos explorar a aplicação prática das operações booleanas na linguagem C por meio de 2 exemplos:

Exemplo 1 – Controle de acesso

ALGORITMOS E TÉCNICAS DE PROGRAMAÇÃO

Imagine um sistema de controle de acesso, em que um usuário precisa apresentar tanto um cartão de identificação válido quanto digitar a senha correta para obter acesso. Em C, isso pode ser representado por meio do código exibido na Figura 1.

```
1 #include <stdio.h>
2
3 int main() {
4     int cartaoValido = 1; // 1 representa verdadeiro
5     int senhaCorreta = 1;
6
7     if (cartaoValido && senhaCorreta) {
8         printf("Acesso concedido.\n");
9     } else {
10        printf("Acesso negado.\n");
11    }
12
13    return 0;
14 }
```

Figura 2 | Controle de acesso.

As variáveis lógicas em C são fundamentais para a lógica condicional em programação. Com elas, os desenvolvedores podem criar algoritmos que respondem dinamicamente às condições do ambiente, permitindo a construção de programas mais flexíveis e adaptáveis. Entender o uso correto dessas variáveis é fundamental para o desenvolvimento eficiente e preciso de lógica de programação em C.

Aplicação operações e expressões em Linguagem C

Neste contexto, exploraremos como as expressões são fundamentais para realizar operações e como as aplicações práticas moldam o cenário do desenvolvimento de software. A eficiência é uma característica da linguagem C.

Essa linguagem foi projetada para fornecer controle direto sobre o hardware do computador, permitindo que os programadores otimizem seus códigos para obter desempenho máximo. As expressões, que consistem em combinações de operadores e operandos, são a base dessa eficiência, permitindo realizar operações matemáticas e lógicas de forma concisa e precisa.

ALGORITMOS E TÉCNICAS DE PROGRAMAÇÃO

As expressões em C vão além das operações aritméticas básicas. Elas incorporam operadores relacionais e lógicos, possibilitando a criação de condições e estruturas de controle. Essa versatilidade serve para desenvolver algoritmos complexos e tomar decisões dinâmicas no fluxo de um programa.

As aplicações práticas das expressões são evidenciadas nas estruturas de controle de C. Seja utilizando condicionais como *if* e *else*, ou estruturas de repetição como *for* e *while*, as expressões determinam o fluxo de execução do programa. Com elas, os desenvolvedores podem criar algoritmos capazes de se adaptar dinamicamente a diferentes situações.

A manipulação eficiente de dados é um dos pilares da programação em C. As expressões são essenciais para operações com variáveis e estruturas de dados. Seja realizando operações matemáticas em um conjunto de dados ou manipulando *strings*, a habilidade de criar expressões concisas é fundamental para a eficácia do código.

Alguns exemplos que representam as expressões em C serão apresentados na sequência. Observe um algoritmo simples que efetua a soma de dois números e apresenta a soma na tela na Figura 3.

ALGORITMOS E TÉCNICAS DE PROGRAMAÇÃO

```
1 #include <stdio.h>
2
3 √ int main() {
4
5     int a, b, soma;
6
7     printf("Digite dois números: ");
8     scanf("%d %d", &a, &b);
9
10    soma = a + b;
11
12    printf("A soma é: %d\n", soma);
13
14    return 0;
15
16 }
```

Figura 3 | Calculadora Simples.

Observe a alocação de memória na Figura 4 por meio da utilização da utilização dos ponteiros.

ALGORITMOS E TÉCNICAS DE PROGRAMAÇÃO

```
1 #include <stdio.h>
2
3 int main() {
4
5     int *ponteiro;
6     ponteiro = (int *)malloc(sizeof(int) * 10);
7
8     return 0;
9
10 }
```

Figura 4 | Alocação dinâmica de memória.

Observe o código a seguir que representa também a manipulação de *string* por meio da Linguagem de Programação, na Figura 5.

```
1
2     char str1[] = "Hello";
3
4     char str2[] = " World!";
5
6     strcat(str1, str2);
7
8     I
```

Figura 5 | Manipulação de strings.

ALGORITMOS E TÉCNICAS DE PROGRAMAÇÃO

Expressões são essenciais para estruturas de controle como *if, else, switch*, na Figura 6.

```
1 < if (idade >= 18) {  
2  
3     printf("É maior de idade.\n");  
4  
5 < } else {  
6  
7     printf("É menor de idade.\n");  
8  
9 }
```

Figura 6 | Fluxo de controle.

As expressões em Linguagem C, fazem parte do esqueleto de qualquer programa, capacitando desenvolvedores a realizar operações matemáticas, criar estruturas condicionais e manipular dados.

As aplicações práticas, desde simples cálculos até alocação dinâmica de memória, destacam a versatilidade dessa linguagem. Dominar expressões é fundamental para criar software eficiente e robusto, proporcionando aos desenvolvedores as ferramentas necessárias para enfrentar desafios complexos no desenvolvimento de sistemas.

Siga em Frente...

Funções predefinidas para Linguagem de Programação

As funções predefinidas, também conhecidas como funções *built-in* ou funções da biblioteca, desempenham um papel fundamental no desenvolvimento de software em linguagens de programação como C.

Essas funções fornecem um conjunto de operações prontas para uso, permitindo que os programadores realizem tarefas complexas com facilidade e eficiência. Neste tópico, vamos

ALGORITMOS E TÉCNICAS DE PROGRAMAÇÃO

explorar o conceito de funções predefinidas, sua importância e exemplos práticos de como utilizá-las na linguagem C.

Funções predefinidas são conjuntos de instruções que executam uma tarefa específica e são disponibilizadas por padrão nas bibliotecas de uma linguagem de programação. Na linguagem C, essas funções estão contidas em diferentes bibliotecas, como `<stdio.h>`, `<stdlib.h>`, `<math.h>`, entre outras. Elas são projetadas para simplificar o desenvolvimento de programas, fornecendo funcionalidades prontas para uso.

Entre os vários fatores que estão situados dentro das funções predefinidas, estão a sua importância. A utilização de funções predefinidas oferece uma série de vantagens para os programadores. Em primeiro lugar, elas economizam tempo e esforço, pois permitem a realização de tarefas complexas com apenas uma chamada de função.

Além disso, as funções predefinidas são amplamente testadas e otimizadas, garantindo sua eficiência e confiabilidade. Isso permite que os desenvolvedores foquem em resolver problemas específicos sem se preocupar com a implementação detalhada de cada função.

Exemplos de funções predefinidas em C

- **Funções de entrada e saída:** a biblioteca `<stdio.h>` contém funções como `printf()` e `scanf()`, que são amplamente utilizadas para entrada e saída de dados no console.
- **Funções matemáticas:** a biblioteca `<math.h>` oferece funções matemáticas como `sqrt()`, `sin()`, `cos()`, `pow()`, entre outras, que permitem realizar operações matemáticas complexas com facilidade.
- **Funções de manipulação de *strings*:** a biblioteca `<string.h>` fornece funções como `strlen()`, `strcpy()`, `strcat()`, que são usadas para manipular strings em C.
- **Funções de alocação de memória:** a biblioteca `<stdlib.h>` inclui funções como `malloc()`, `calloc()`, `realloc()`, que são utilizadas para alocar e liberar memória dinamicamente.

Vamos a um exemplo de utilização envolvendo algumas dessas bibliotecas, como é observado na Figura 7.

Vamos Exercitar?

Descrição da situação-problema:

Você é responsável pela logística de embarque de um voo internacional. Diversos passageiros precisam ser acomodados em diferentes classes e posições do avião. O desafio é otimizar o espaço disponível, considerando as diferentes configurações de assentos em cada classe (econômica, executiva e primeira classe).

ALGORITMOS E TÉCNICAS DE PROGRAMAÇÃO

A aeronave tem capacidades diferentes em cada classe, e você precisa distribuir os passageiros de forma eficiente, maximizando o número de assentos ocupados, respeitando as características específicas de cada classe. Além disso, é necessário considerar requisitos especiais, como passageiros que viajam com crianças, necessidades de acessibilidade, entre outros.

Resolução da situação-problema:

Identificação das Classes e Capacidades:

- Determine o número de assentos disponíveis em cada classe.
- Considere as configurações específicas de assentos em cada classe.

Registro dos Passageiros:

- Registre o número total de passageiros em cada classe.
- Anote informações adicionais, como necessidades especiais.

Cálculos Matemáticos:

- Utilize operações matemáticas para otimizar a distribuição de passageiros.
- Considere estratégias para maximizar o preenchimento dos assentos.

Acomodação Eficiente:

- Desenvolva um algoritmo que organize os passageiros em assentos disponíveis de maneira eficiente.
- Leve em conta requisitos especiais e preferências dos passageiros sempre que possível.

Validação e Ajustes:

- Verifique se todas as restrições foram atendidas.
- Faça ajustes conforme necessário para otimizar ainda mais a distribuição.

A aplicação de cálculos matemáticos por meio de operações algorítmicas permitiu otimizar o processo de embarque do voo, garantindo uma distribuição eficiente dos passageiros em diferentes classes.

A abordagem sistemática de utilizar algoritmos para resolver desafios logísticos proporciona não apenas eficiência operacional, mas também uma experiência mais confortável e personalizada para os passageiros. A aplicação prática de conceitos matemáticos na resolução desse problema destaca a importância da computação e algoritmos na otimização de processos do mundo real.

ALGORITMOS E TÉCNICAS DE PROGRAMAÇÃO

Saiba mais

Para saber mais sobre os conceitos e formas de aplicação dos "Operadores e Expressões" na linguagem de programação C e quais suas características, acesse em sua Biblioteca Virtual o livro [Elementos de Programação em C](#), Capítulo 6 - páginas 112 a 144.

Referências

CORMEN, T. **Algoritmos – Teoria e Prática**. 3.ed. Rio de Janeiro: LTC, 2022.

CORMEN, T. **Desmistificando Algoritmos**. Grupo GEN, 2013.

FERREIRA, A. B. H. **Novo Aurélio Século XXI: o dicionário da língua portuguesa**. 3 ed. totalmente rev. e ampl. Rio de Janeiro: Nova Fronteira, 1999.

MANZANO, J. A. N G.; OLIVEIRA, J. F. **Estudo Dirigido de Algoritmos**. 15. ed. Editora Saraiva, 2012.

MENÉNDEZ, A. **Simplificando Algoritmos**. Grupo GEN, 2023.

PINHEIRO, F. A. C. **Elementos de Programação em C**. Porto Alegre: Bookman, 2012.

SERPA, M. S.; RODRIGUES, T. N.; ALVES, Í. C.; *et al.* **Análise de Algoritmos**. Grupo A, 2021.

Aula 4

Escopo de Variáveis

Escopo de variáveis

ALGORITMOS E TÉCNICAS DE PROGRAMAÇÃO



Este conteúdo é um vídeo!

Para assistir este conteúdo é necessário que você acesse o AVA pelo computador ou pelo aplicativo. Você pode baixar os vídeos direto no aplicativo para assistir mesmo sem conexão à internet.

Dica para você

Aproveite o acesso para baixar os slides do vídeo, isso pode deixar sua aprendizagem ainda mais completa.

Olá, estudante! Nesta videoaula, exploraremos conceitos fundamentais, como operadores lógicos e de comparação, técnicas eficazes de conversão de tipos de dados e estratégias para entrada e saída de dados. Estes são fatores importantes para que você domine a arte da programação e alcance resultados mais sólidos em seus projetos.

Não perca a chance de enriquecer seus conhecimentos e expandir sua compreensão sobre algoritmos. Convidamos você a se juntar a nós nessa cativante jornada de descobertas, onde fortaleceremos suas habilidades nesta área vital. Vamos em frente!

Ponto de Partida

Olá, estudante! Seja bem-vindo a esta aula, em que adentraremos em um novo território de conhecimento, expandindo sua jornada de aprendizado.

Mergulharemos em diversos tipos de variáveis presentes na Linguagem de Programação, buscando compreender como esses elementos contribuem para a construção de algoritmos e programas de computadores.

O foco principal desta aula recai sobre a exploração de conceitos-chave relacionados a Operadores lógicos e de comparação, técnicas de conversão de tipos de dados, bem como estratégias eficazes de entrada e saída de dados. Estes são pilares essenciais para a manipulação inteligente de informações em linguagem de programação.

Destacaremos, especialmente, os "operadores lógicos e de comparação", mostrando como eles desempenham um papel importante na lógica de programação. Demonstraremos, através de exemplos práticos, como utilizar esses operadores para aprimorar a eficiência dos seus algoritmos.

Diante disso, será apresentado um programa que tem o objetivo de executar instruções de entrada e saída de dados considerando a construção de algoritmos e operações necessárias para resolver o problema proposto.

ALGORITMOS E TÉCNICAS DE PROGRAMAÇÃO

Convidamos você a mergulhar nestes conteúdos fundamentais, essenciais para a construção sólida de sua base técnica em desenvolvimento de sistemas. Vamos lá?

Vamos Começar!

Na programação de computadores, a compreensão e domínio de conceitos fundamentais são essenciais para o desenvolvimento de códigos eficientes e robustos. Neste contexto, três temas merecem destaque: operadores lógicos e comparação, conversão de tipos de dados, e entrada e saída de dados. Cada um desses tópicos desempenha um papel vital na construção de algoritmos e na interação eficaz entre o programa e o usuário.

Compreender como esses operadores funcionam e como utilizá-los de maneira eficiente é crucial para criar algoritmos lógicos e eficazes. Através de uma abordagem prática, exploraremos como essas ferramentas podem ser aplicadas para criar fluxos de controle condicionais, contribuindo para a flexibilidade e adaptabilidade dos programas.

A conversão de tipos de dados é outra habilidade indispensável, uma vez que muitas linguagens de programação requerem consistência na manipulação de diferentes tipos de variáveis.

Entender como converter dados de um tipo para outro não apenas previne erros de execução, mas também permite a utilização otimizada dos recursos do sistema. Discutiremos técnicas de conversão que garantem a coerência dos dados, proporcionando uma base sólida para o desenvolvimento de algoritmos mais robustos e eficientes.

Por fim, a interação entre o programa e o usuário é facilitada pela entrada e saída de dados. Assim, compreender como receber informações do usuário e apresentar resultados de maneira clara e amigável é importante para a usabilidade de qualquer aplicação.

Exploraremos métodos de entrada que tornam a interação intuitiva e métodos de saída que comunicam eficientemente os resultados, promovendo a eficácia e a usabilidade dos programas desenvolvidos. Esses conceitos são peças fundamentais no quebra-cabeça da programação, capacitando os desenvolvedores a criarem aplicações mais interativas e eficientes.

Operadores lógicos e comparação

Os operadores lógicos e de comparação são ferramentas fundamentais para avaliar condições dentro de um código, permitindo a tomada de decisões com base em comparações de valores.

Operadores de comparação

ALGORITMOS E TÉCNICAS DE PROGRAMAÇÃO

No universo do Portugol, os operadores de comparação são cruciais para avaliar condições e realizar verificações em tempo de execução. Essas ferramentas permitem comparar valores e determinar relações entre eles. Por exemplo, ao usar o operador `>`, podemos verificar se um número é maior que outro.

Isso é fundamental para criar algoritmos que respondam de maneira dinâmica a diferentes situações, como verificar se a idade de um usuário é maior que 18 para determinar se é elegível para certas ações. Exemplo:

```

1  se idade > 18 então
2      escreva("Usuário é maior de idade.")
3  senão
4      escreva("Usuário é menor de idade.")
5  fimse
6

```

Figura 1 | Exemplo de comparação.

Operadores lógicos

Além dos operadores de comparação, os operadores lógicos são peças-chave na construção de lógica de programação mais complexa. Com E (AND), OU (OR) e NÃO (NOT), podemos combinar condições de maneira sofisticada. Isso é essencial para criar algoritmos que tomam decisões baseadas em múltiplas condições, como verificar se uma pessoa é maior de idade E possui uma carteira de habilitação.

```

1  ~ se (idade >= 18) E (temCarteiraDeHabilitacao == verdadeiro) então
2      escreva("Pode dirigir.")
3  ~ senão
4      escreva("Não pode dirigir.")
5  fimse

```

Figura 2 | Exemplo de operador lógico.

Combinação de operadores

A verdadeira potência emerge quando combinamos operadores de comparação e lógicos. Essa sinergia nos permite criar condições complexas que se adaptam a uma variedade de cenários.

ALGORITMOS E TÉCNICAS DE PROGRAMAÇÃO

Por exemplo, ao usar uma combinação de E e OU, podemos verificar se uma pessoa é maior de idade E possui carteira de habilitação OU tem permissão dos pais. Veja o exemplo:

```

1  se (idade >= 18) E ((temCarteiraDeHabilitacao == verdadeiro) OU (temPermissaoDosPais == verdadeiro)) então
2    escreva("Pode dirigir.")
3  senão
4    escreva("Não pode dirigir.")
5  fimse

```

Figura 3 | Operadores lógicos e de comparação.

Ao entender esses operadores, podemos refinar nossas tomadas de decisão, tornando nossos programas mais flexíveis e adaptáveis. Essa capacidade é crucial para criar algoritmos robustos que lidam eficientemente com diversas situações, resultando em código mais legível, eficiente e fácil de manter.

Conversão de tipos de dados

A conversão de tipos de dados é um aspecto crucial em programação, especialmente quando lidamos com diferentes tipos de variáveis. Em Portugol, a capacidade de converter dados de um tipo para outro é essencial para garantir que as operações sejam realizadas de maneira consistente. Como exemplo, podemos converter um número inteiro para ponto flutuante ou uma string para um número, adaptando os dados conforme necessário.

Conversão implícita e explícita

Em muitos casos, a linguagem de programação realiza automaticamente a conversão de tipos, conhecida como conversão implícita. No entanto, há momentos em que precisamos realizar a conversão explicitamente, especificando o tipo desejado. Ao concatenar uma string com um número, a conversão implícita pode ocorrer automaticamente, mas, se necessário, podemos realizar uma conversão explícita.

```

1 // Conversão implícita
2 inteiro idade = 25
3 texto mensagem = "A idade é: " + idade // Conversão automática de inteiro para texto
4
5 // Conversão explícita
6 real preco = 50.75
7 inteiro precoInteiro = inteiro(preco) // Conversão explícita de ponto flutuante para inteiro

```

Figura 4 | Tipos de Conversão.

Tratamento de erros na conversão

ALGORITMOS E TÉCNICAS DE PROGRAMAÇÃO

É importante estar ciente de possíveis problemas durante a conversão de tipos, como a perda de dados ao converter um número decimal para inteiro. Algumas linguagens, incluindo Portugol, podem gerar erros se a conversão não for possível ou resultar em perda de informação. Portanto, é prudente realizar verificações adequadas antes de converter tipos para evitar possíveis falhas no programa.

Ao utilizar a conversão de tipos, é fundamental entender as nuances específicas da linguagem que estamos utilizando. Além disso, devemos considerar o impacto da conversão nos dados e garantir que o resultado atenda às expectativas do programa. Ao aplicar a conversão de tipos de maneira consciente, podemos melhorar a flexibilidade e a eficiência de nossos algoritmos.

Este desenvolvimento abrangente explora a importância da conversão de tipos de dados em Portugol, destacando tanto os aspectos fundamentais quanto as considerações práticas ao lidar com diferentes tipos de variáveis.

Siga em Frente...

Entrada e saída de dados

A entrada de dados desempenha um papel crucial na interação entre o usuário e o programa. Em Portugol, utilizamos comandos específicos para capturar informações fornecidas pelo usuário. O comando `leia` é fundamental para essa tarefa, permitindo que o programa solicite e receba dados do usuário durante a execução. Por exemplo, podemos usar o `leia` para obter o nome de uma pessoa a partir da entrada do teclado.

```
1  texto nome
2
3  escreva("Digite seu nome: ")
4
5  leia(nome)
6
```

Figura 5 | Entrada.

ALGORITMOS E TÉCNICAS DE PROGRAMAÇÃO

A saída de dados é a forma como o programa apresenta resultados ao usuário. O comando escreva é amplamente utilizado em Portugol para exibir informações na tela. Podemos imprimir variáveis, textos ou resultados de operações, tornando a saída de dados uma ferramenta poderosa para comunicar informações ao usuário.

```
1  inteiro idade = 30
2  escreva("Sua idade é: ", idade)
3
```

Figura 6 | Saída.

Além de simplesmente exibir dados, a formatação adequada é essencial para tornar a saída comprehensível e esteticamente agradável. Em Portugol, podemos utilizar sequências de escape, como "\n" para quebras de linha, ou "\t" para tabulações, aprimorando a apresentação dos resultados.

Após capturar dados de entrada, muitas vezes precisamos convertê-los para o tipo adequado antes de utilizá-los no programa. Isso envolve a combinação de comandos de entrada e conversão de tipos, como exemplificado abaixo, onde capturamos um número como texto e o convertemos para inteiro.

```
1  texto numeroTexto
2  inteiro numero
3
4  escreva("Digite um número: ")
5  leia(numeroTexto)
6
7  ~ se (inteiro(numeroTexto, numero)) então
8    |   escreva("Número digitado: ", numero)
9  ~ senão
10 |   escreva("Por favor, digite um número válido.")
11 fimse
```

Figura 7 | Manipulação de dados.

Ao explorarmos a entrada e saída de dados em Portugol, destacamos a importância vital desses processos na interação eficaz entre o usuário e o programa. A capacidade de receber dados do

ALGORITMOS E TÉCNICAS DE PROGRAMAÇÃO

usuário por meio do comando `leia` proporciona flexibilidade, permitindo a personalização e adaptação dos algoritmos conforme as necessidades específicas de cada execução.

Da mesma forma, a saída de dados, facilitada pelo comando `escreva`, é essencial para comunicar informações relevantes ao usuário. A possibilidade de formatar a apresentação dos resultados, utilizando sequências de escape e técnicas de formatação, não apenas aprimora a estética, mas também melhora a compreensão dos dados apresentados.

Além disso, ressaltamos a importância da manipulação adequada dos dados de entrada, incluindo a conversão de tipos quando necessário. Essa prática garante a integridade e a coerência dos dados, prevenindo possíveis erros durante a execução do programa.

Em resumo, a entrada e saída de dados não são apenas processos técnicos, mas elementos fundamentais na construção de algoritmos interativos e eficientes. Ao incorporar habilidades de entrada e saída de dados de maneira consciente em nossos programas, podemos criar soluções mais dinâmicas, adaptáveis e acessíveis aos usuários.

Portanto, a compreensão profunda desses conceitos é essencial para qualquer programador que busca desenvolver aplicações robustas e user-friendly em Portugol e em outras linguagens de programação.

Conclusão

Ao abordarmos os tópicos essenciais de Operadores Lógicos e de Comparação, Conversão de Tipos de Dados e Entrada e Saída de Dados em Portugol, percebemos a interconexão crucial desses conceitos no desenvolvimento de algoritmos robustos e interativos.

Os Operadores Lógicos e de Comparação fornecem as ferramentas fundamentais para a tomada de decisões dentro dos programas, permitindo que estes ajam de maneira inteligente diante de diferentes cenários. Essa habilidade é vital para a construção de lógica de programação eficiente e adaptável.

A Conversão de Tipos de Dados, por sua vez, oferece flexibilidade no tratamento de variáveis, permitindo a adaptação de informações conforme necessário. Com a capacidade de converter dados de um tipo para outro, garantimos a consistência e integridade dos dados manipulados pelo programa.

Por fim, a Entrada e Saída de Dados emergem como a interface entre o programa e o usuário, desempenhando um papel crucial na interatividade. Através dos comandos `leia` e `escreva`, podemos capturar informações fornecidas pelo usuário e apresentar resultados de maneira clara e compreensível.

ALGORITMOS E TÉCNICAS DE PROGRAMAÇÃO

Compreender profundamente esses três aspectos: Operadores Lógicos e de Comparação, Conversão de Tipos de Dados e Entrada e Saída de Dados, é essencial para programadores que buscam criar soluções eficazes, interativas e adaptáveis em Portugol.

A habilidade de aplicar esses conceitos de maneira sinérgica capacita o desenvolvedor a construir algoritmos que não apenas executam tarefas, mas também respondem de maneira inteligente a diferentes situações, proporcionando uma experiência de usuário mais dinâmica e satisfatória.

Assim, a maestria desses conceitos representa um passo significativo na jornada de qualquer programador em direção à excelência na arte da programação.

Vamos Exercitar?

Descrição da situação-problema:

Você foi designado para desenvolver um programa em linguagem C para auxiliar no gerenciamento de produtos em um estabelecimento comercial. O programa deve ser capaz de realizar operações de entrada e saída de dados relacionadas aos produtos disponíveis no estoque.

Problema: Desenvolver o cadastro de produtos, consulta de produtos, atualização de estoque e a listagem de produtos.

Resolução da situação-problema:

Cadastro de Produtos:

- Permitir ao usuário cadastrar novos produtos, incluindo informações como nome, preço unitário e quantidade em estoque.

Consulta de Produtos:

- Implementar uma funcionalidade que permita ao usuário consultar informações sobre um produto específico, informando o nome ou código do produto.

Atualização de Estoque:

- Possibilitar a atualização da quantidade em estoque de um produto, considerando as operações de entrada (compras) e saída (vendas).

Listagem de Produtos:

ALGORITMOS E TÉCNICAS DE PROGRAMAÇÃO

- Gerar uma lista completa dos produtos cadastrados, exibindo nome, preço unitário e quantidade em estoque.

A criação desse programa em linguagem C permite uma gestão eficiente do estoque de produtos, facilitando o cadastro, consulta, atualização e listagem. Essa solução é valiosa para estabelecimentos comerciais, proporcionando um controle mais preciso e contribuindo para a tomada de decisões assertivas.

O uso de estruturas de dados e operações básicas de entrada e saída destaca a importância dos fundamentos da programação na resolução de problemas práticos. Esse tipo de aplicação é crucial para ambientes profissionais, onde a organização e o gerenciamento de informações são essenciais para o sucesso do negócio.

O aprendizado desses conceitos permite aos programadores desenvolverem soluções eficientes e alinhadas com as demandas do mercado.

Saiba mais

Para aprofundar seu conhecimento sobre variáveis complexas como "Ponteiros e Vetores" na linguagem de programação C, e quais suas características, acesse a Biblioteca Virtual, o livro [*Elementos de Programação em C*](#), Capítulo 3 - páginas 55 a 61.

Referências

CORMEN, T. **Algoritmos – Teoria e Prática**. 3.ed. Rio de Janeiro: LTC, 2022.

CORMEN, T. **Desmistificando Algoritmos**. Grupo GEN, 2013.

FERREIRA, A. B. H. **Novo Aurélio Século XXI: o dicionário da língua portuguesa**. 3 ed. totalmente rev. e ampl. Rio de Janeiro: Nova Fronteira, 1999.

MANZANO, J. A. N G.; OLIVEIRA, J. F. **Estudo Dirigido de Algoritmos**. 15. ed. Editora Saraiva, 2012.

MENÉNDEZ, A. **Simplificando Algoritmos**. Grupo GEN, 2023.

PINHEIRO, F. A. C. **Elementos de Programação em C**. Porto Alegre: Bookman, 2012.

SERPA, M. S.; RODRIGUES, T. N.; ALVES, Í. C.; *et al.* **Análise de Algoritmos**. Grupo A, 2021.

ALGORITMOS E TÉCNICAS DE PROGRAMAÇÃO

Aula 5

Encerramento da Unidade

Videoaula de Encerramento



Este conteúdo é um vídeo!

Para assistir este conteúdo é necessário que você acesse o AVA pelo computador ou pelo aplicativo. Você pode baixar os vídeos direto no aplicativo para assistir mesmo sem conexão à internet.

Dica para você

Aproveite o acesso para baixar os slides do vídeo, isso pode deixar sua aprendizagem ainda mais completa.

Olá, estudante! Seja bem-vindo à nossa videoaula que vai abordar "Variáveis e Tipos de Dados Compostos Aplicados em Programação". Juntos, iremos estudar os conceitos e características que estão por trás de conteúdos específicos de algoritmos. Nessa videoaula, mergulharemos nos fundamentos desse tipo de variável abordando sua aplicabilidade na criação de soluções algorítmicas.

Então, não perca tempo! Inicie a aula, aproveite e embarque em mais este aprendizado consistente no cenário de algoritmos. Assista agora e dê mais um passo em busca do conhecimento.

Ponto de Chegada

Olá estudante! Para desenvolver a competência desta Unidade, que é "Identificar as constantes e variáveis em linguagens de programação", você deverá primeiramente conhecer os tipos de dados primitivos antes de adentrar em um outro hemisfério com tipos mais complexos e versáteis que também são utilizados no mundo da programação.

Você vai conhecer e compreender onde esses tipos são utilizados, qual a sua necessidade e aplicação no cenário profissional.

Aprender sobre tipos de dados como vetores, matrizes e ponteiros é como desbloquear portas para um mundo vasto e eficiente na programação. Esses conceitos não são apenas blocos de

ALGORITMOS E TÉCNICAS DE PROGRAMAÇÃO

construção, mas ferramentas poderosas que aprimoram a capacidade de resolver problemas de maneira eficaz.

Entre os benefícios podemos destacar:

- O desenvolvimento da habilidade de organizar dados de forma estruturada quando se estuda os conceitos aplicações dos *Arrays*, conhecido como vetores e matrizes.
- Os ponteiros que vão oportunizar a manipulação direta do endereçamento de memória que ele está alocado. Essa prática pode evitar desperdícios no tempo de execução do programa tornando os algoritmos mais ágeis.

Em resumo, compreender como esses "tipos" peculiares de dados trabalham é um investimento na eficiência e versatilidade na criação de soluções algorítmicas mais robustas e com melhor desempenho.

É Hora de Praticar!

Em uma loja de eletrodomésticos que vende diversos produtos, um cliente comprou um liquidificador recentemente e enfrentou diversos problemas com o produto, desde vazamentos até dificuldades na operação. Insatisfeito, ele decide acionar a garantia para resolver a situação. O "desafio" é utilizar um *Array* (vetor) como estratégia de armazenagem das reclamações sobre cada produto, pois a chance de ser mais do que um problema é grande. Criando um vetor no algoritmo você consegue em uma única variável armazenar várias informações ao mesmo tempo.

A sugestão de passos para o registro e solução do(s) problema(s) do produto em questão seriam:

- Registro Detalhado.
 - Análise do Problema.
 - Verificação da Garantia.
 - Priorização e Organização.
 - Opções de Resolução.
 - Comunicação com o Cliente.
 - Execução da Solução Escolhida.
 - Feedback do Cliente.
-
- Já pensou em vetores e matrizes como arquitetos da organização de dados em algoritmos? Assim como arrumamos nossas gavetas para facilitar a busca de itens específicos, esses conceitos estruturam dados de forma eficiente.
 - No entanto, vale refletir: a verdadeira eficácia está na estruturação desses dados ou na habilidade de manipulá-los com inteligência? A organização por si só é suficiente para criar algoritmos poderosos, ou é a maneira como interagimos com essas estruturas que define nossa maestria na programação?

ALGORITMOS E TÉCNICAS DE PROGRAMAÇÃO

- Imagine que ponteiros de memória são como guias secretos, apontando diretamente para informações relevantes no vasto território da memória do computador. No entanto, a reflexão vai além: como essa capacidade de apontar diretamente para endereços de memória influencia a eficiência de um algoritmo? Será que somente isso é o bastante?
- A agilidade está na habilidade de navegar por esses caminhos ou na responsabilidade de não se perder nesse confuso labirinto de dados? Em um sentido mais amplo, será que os ponteiros nos mostram o caminho, mas a verdadeira maestria está em saber para onde direcioná-los?

Reflita sobre essas questões e continue formando o seu senso crítico, buscando sempre o equilíbrio entre o aprendizado para formar o seu conhecimento e a aceitação de todo conteúdo como uma rasa verdade.

Sugestão de solução

Passo 1: Registro detalhado:

- O cliente registra uma reclamação detalhada, incluindo vazamentos, dificuldades na operação e qualquer outro problema. Esses itens são armazenados em um vetor de reclamação. Como sugestão você pode criar dentro da ferramenta um vetor para armazenar essa reclamação. Exemplo do nome do vetor: "vetReclame[]". E depois disso fazer alguns testes no algoritmo.

Passo 2: Análise do problema:

- A equipe de suporte técnico analisa cada item registrado na reclamação, verificando se estão cobertos pela garantia.

Passo 3: Verificação da garantia:

- É necessário verificar se o produto está dentro do período de garantia e se os problemas relatados estão cobertos pelos termos da garantia.

Passo 4: Priorização e Organização:

- Os itens registrados são priorizados e organizados em um vetor de itens, destacando a gravidade e a frequência de cada problema.

Passo 4: Priorização e organização:

- Os itens registrados são priorizados e organizados em um vetor de itens, destacando a gravidade e a frequência de cada problema.

Passo 5: Opções de resolução:

- Com os itens organizados, a equipe avalia as opções de resolução para cada problema. Pode incluir reparo, substituição ou reembolso, dependendo da gravidade de cada item.

ALGORITMOS E TÉCNICAS DE PROGRAMAÇÃO

Passo 6: Comunicação com o cliente:

- A equipe entra em contato com o cliente, apresenta a análise dos problemas e discute as opções de resolução para cada item registrado.

Passo 7: Execução da solução escolhida:

- Após a escolha do cliente para cada item, a equipe executa as soluções acordadas. Se for um reparo, múltiplas tarefas são realizadas de maneira eficiente.

Passo 8: Feedback do cliente:

- Após a resolução de cada problema, a equipe solicita feedback ao cliente para avaliar a satisfação com o processo de resolução de cada item.

Conclusão

O estudo de caso destaca a complexidade da resolução de múltiplos problemas em um único produto sob garantia. A utilização de vetores facilita a organização e priorização desses problemas, proporcionando uma abordagem estruturada para a resolução eficaz.

A análise crítica e a comunicação transparente com o cliente são fundamentais para assegurar a eficiência do processo e garantir a satisfação do cliente em relação a cada item reclamado.

Neste infográfico, apresentamos resumidamente os principais conceitos relacionados a: Aplicação de constantes, variáveis, operações e escopo de programação. Todos esses, tópicos fundamentais para a disciplina de Algoritmos e Técnicas de Programação.

Convidamos você estudante, a refletir sobre os conceitos apresentados a seguir:

ALGORITMOS E TÉCNICAS DE PROGRAMAÇÃO



Figura | Elementos de dados complexos.

CORMEN, Thomas. Algoritmos – Teoria e Prática. 3.ed. Rio de Janeiro: LTC, 2022.

ALGORITMOS E TÉCNICAS DE PROGRAMAÇÃO

CORMEN, T. **Desmistificando Algoritmos.** [S. I.]: Grupo GEN, 2013.

MANZANO, J. A. N. G.; OLIVEIRA, J. F. **Estudo Dirigido de Algoritmos.** [S. I.]: Editora Saraiva, 1997.

MENÉNDEZ, A. **Simplificando Algoritmos.** [S. I.]: Grupo GEN, 2023.

SERPA, M. S.; et al. **Análise de Algoritmos.** [S. I.]: Grupo A, 2021.

Unidade 3

Estruturas de Decisão e Repetição

Aula 1

Estruturas de Decisão Condisional (Se)

Estruturas de decisão condicional (Se)

Este conteúdo é um vídeo!



Para assistir este conteúdo é necessário que você acesse o AVA pelo computador ou pelo aplicativo. Você pode baixar os vídeos direto no aplicativo para assistir mesmo sem conexão à internet.

Dica para você

Aproveite o acesso para baixar os slides do vídeo, isso pode deixar sua aprendizagem ainda mais completa.

Olá, estudante!

Nesta aula abordaremos assuntos relativos à algoritmos e técnicas de programação como: Desvio condicional simples – Se / Fim-se, Desvio condicional composto – Se / Senão / Fim-se e Desvio condicional encadeado – testes condicionais encadeados.

Para que você comprehenda os conteúdos da melhor forma possível, vamos trazer exemplos em que esses conteúdos podem ser aplicados na prática, tornando seu aprendizado mais significativo.

Os conhecimentos adquiridos aqui não apenas expandem a sua compreensão do assunto, mas também fortalecem a sua capacidade de pensamento crítico e resolução de problemas. Aproveite cada oportunidade e cada leitura para explorar os conceitos apresentados, participando ativamente na construção do seu conhecimento.

ALGORITMOS E TÉCNICAS DE PROGRAMAÇÃO

Ponto de Partida

Olá, estudante. Desejamos boas-vindas à esta aula e espero que esse conteúdo realmente agregue valor no seu aprendizado.

Hoje, vamos explorar conteúdos referentes a algoritmos e técnicas de programação e teremos como foco o estudo nos diferentes tipos de desvios condicionais.

A compreensão dos conceitos de desvio condicional é fundamental para qualquer estudante que deseje aprofundar seus conhecimentos em programação.

Inicialmente, o estudo do desvio condicional simples, utilizando a estrutura "se/fim-se", é essencial para entender como um programa pode tomar diferentes caminhos de execução com base em condições específicas. Dominar esse conceito permitirá a você criar programas mais dinâmicos e adaptáveis, capazes de responder de maneira inteligente a diferentes cenários.

Além disso, o desvio condicional composto, que incorpora a estrutura "se/senão/fim-se", ampliará ainda mais a sua capacidade de desenvolver algoritmos mais usuais. Compreender como utilizar esse tipo de desvio condicional é importante para lidar com situações em que há mais de uma possibilidade de execução, possibilitando a criação de programas mais eficientes.

Por fim, o estudo do desvio condicional encadeado, que envolve testes condicionais encadeados, é fundamental para lidar com situações complexas em que múltiplas condições precisam ser avaliadas sequencialmente.

Dominar essa técnica permitirá você a desenvolver algoritmos mais complexos e completos, capazes de lidar com uma ampla gama de situações do mundo real. Portanto, dedicar tempo e esforço ao estudo desses três conteúdos será fundamental para que você adquira habilidades e se destaque na área de programação e desenvolvimento de software.

Diante disso, será apresentado o desenvolvimento de um algoritmo que tem o objetivo de controlar uma nave espacial em uma missão de pesquisa na Lua.

Convidamos você a iniciar a construção de uma base sólida para sua jornada no universo da programação. Preparado para essa imersão?

Vamos Começar!

Desvio condicional simples – Se / Fim-se

O desvio condicional simples utilizando a estrutura "se/fim-se" é um dos conceitos fundamentais da programação, desempenhando um importante papel na criação de algoritmos e na resolução

ALGORITMOS E TÉCNICAS DE PROGRAMAÇÃO

de problemas computacionais.

Essa estrutura condicional permite que um programa tome decisões com base em condições específicas, desviando o fluxo de execução conforme necessário, uma vez que a capacidade de executar ações com base em diferentes cenários é essencial para a criação de software dinâmico e adaptável.

Ao compreender e aplicar o desvio condicional simples, os programadores podem desenvolver algoritmos que respondam a uma variedade de situações, tornando seus programas mais úteis e eficientes. Desde a simples validação de entrada de dados até a implementação de lógica de negócios complexa, a utilização do "se/fim-se" permite aos desenvolvedores controlar o fluxo de execução de seus programas de forma precisa e flexível.

Além disso, o desvio condicional simples é amplamente aplicável em uma variedade de contextos, desde a programação de pequenos scripts até o desenvolvimento de sistemas de software complexos. Seja na automação de tarefas do dia a dia ou na implementação de algoritmos avançados, a compreensão deste conceito é fundamental para qualquer programador que deseje criar soluções eficientes.

A seguir veremos três exemplos em que esse conceito pode ser aplicado para que você compreenda melhor como ele funciona na prática. Observe:

1. Verificação de idade: para criar um programa em C para verificar se um usuário é maior de idade, é possível usar um desvio condicional simples para comparar a idade fornecida pelo usuário com o limite de idade para ser considerado maior de idade.

Se a idade for maior ou igual a 18, o programa exibirá uma mensagem informando que o usuário é maior de idade. Exemplo de código em C: Observe na Figura 1 um exemplo de verificação de idade na Linguagem C:

ALGORITMOS E TÉCNICAS DE PROGRAMAÇÃO

```
1 #include <stdio.h>
2
3 int main() {
4     int idade;
5     printf("Digite sua idade: ");
6     scanf("%d", &idade);
7
8     if (idade >= 18) {
9         printf("Você é maior de idade.\n");
10    }
11
12    return 0;
13 }
```

Figura 1 | Exemplo de verificação de idade em Linguagem C.

2. Verificação Par ou Ímpar: para criar um programa em C para verificar se um número é par ou ímpar, pode-se usar um desvio condicional simples para verificar se o resto da divisão do número por 2 é igual a zero. Se for, o número é par; caso contrário, é ímpar. Vejamos na Figura 2 um exemplo de verificação Par ou Ímpar em C:

ALGORITMOS E TÉCNICAS DE PROGRAMAÇÃO

```
1 #include <stdio.h>
2
3 int main() {
4     int numero;
5     printf("Digite um número: ");
6     scanf("%d", &numero);
7
8     if (numero % 2 == 0) {
9         printf("O número é par.\n");
10    } else {
11        printf("O número é ímpar.\n");
12    }
13
14    return 0;
15 }
```

Figura 2 | Exemplo de verificação Par ou Ímpar em C.

3. Verificação de Número Positivo: para criar um programa em C para verificar se um número fornecido pelo usuário é positivo, pode-se usar um desvio condicional simples para verificar se o número é maior que zero. Se for, o programa exibirá uma mensagem informando que o número é positivo. Observe na figura 3 um exemplo de verificação de número positivo em C:

ALGORITMOS E TÉCNICAS DE PROGRAMAÇÃO

```
1 #include <stdio.h>
2
3 ∵ int main() {
4     int numero;
5     printf("Digite um número: ");
6     scanf("%d", &numero);
7
8     ∵ if (numero > 0) {
9         printf("O número é positivo.\n");
10    }
11
12    return 0;
13 }
14 }
```

Figura 3 | Exemplo de verificação de número positivo em C.

Vale lembrar que se o número informado for negativo ou 0 (zero), o programa não entrará dentro da condição, portanto, nada é apresentado na tela.

Esses exemplos ilustram como o desvio condicional simples "se/fim-se" é utilizado na programação para tomar decisões com base em condições específicas, controlando o fluxo de execução do programa de acordo com os dados fornecidos ou condições estabelecidas.

Desvio condicional composto – Se / Senão / Fim-se

O desvio condicional composto, utilizando a estrutura "se/senão/fim-se", é muito utilizado na programação. Esta estrutura permite que um programa tome decisões com base em duas possibilidades distintas, proporcionando uma maior flexibilidade e capacidade de resposta a diferentes cenários. Sua importância na programação reside na capacidade de lidar não apenas com uma única condição, mas também com sua negação, ampliando assim as possibilidades de controle de fluxo.

ALGORITMOS E TÉCNICAS DE PROGRAMAÇÃO

Com a aplicação do desvio condicional composto, os programadores podem desenvolver algoritmos capazes de lidar de forma eficaz com uma variedade de situações em que se deparam, desde a implementação de lógica de negócios em sistemas de software até a criação de jogos interativos, a utilização do "se/senão/fim-se" permite aos desenvolvedores controlar o fluxo de execução de seus programas de maneira mais precisa e abrangente.

O desvio condicional composto é amplamente aplicável em diversas áreas da programação, desde a construção de interfaces de usuário até o processamento de dados complexos.

Seja na criação de aplicativos móveis que se adaptam às preferências do usuário ou na implementação de algoritmos de otimização em sistemas de inteligência artificial, a compreensão deste conceito é fundamental para qualquer programador que deseje criar diferentes soluções tecnológicas.

A seguir veremos três exemplos em que esse conceito pode ser aplicado para que você compreenda melhor como ele funciona na prática. Observe:

1. Verificação de Maioridade: neste exemplo, o programa solicita ao usuário que insira sua idade e verifica se ele é maior de idade (idade igual ou superior a 18 anos). Se a idade for maior ou igual a 18, o programa exibe uma mensagem indicando que o usuário é maior de idade. Caso contrário, exibe uma mensagem informando que o usuário é menor de idade. Observe na Figura 4 um exemplo de verificação de Maioridade em pseudocódigo:

ALGORITMOS E TÉCNICAS DE PROGRAMAÇÃO

```
1 #include <stdio.h>
2
3 √ int main() {
4     int idade;
5
6     printf("Digite sua idade: ");
7     scanf("%d", &idade);
8
9     √ if (idade >= 18) {
10         printf("Você é maior de idade.\n");
11     } else {
12         printf("Você é menor de idade.\n");
13     }
14
15     return 0;
16 }
```

Figura 4 | Exemplo de controle de verificação de login em C.

2. Classificação de Números: esse exemplo vai demonstrar como o desvio condicional composto permite que o programa tome diferentes caminhos de execução com base em diferentes condições, classificando um número como positivo, negativo ou zero de acordo com o valor fornecido pelo usuário. Observe na Figura 5 um exemplo de Classificação de Números na linguagem C:

ALGORITMOS E TÉCNICAS DE PROGRAMAÇÃO

```
1 #include <stdio.h>
2
3 int main() {
4     int numero;
5
6     printf("Digite um número: ");
7     scanf("%d", &numero);
8
9     if (numero > 0) {
10         printf("O número é positivo.\n");
11     } else if (numero < 0) {
12         printf("O número é negativo.\n");
13     } else {
14         printf("O número é zero.\n");
15     }
16
17     return 0;
18 }
```

Figura 5 | Exemplo de classificação de números em C.

Neste exemplo:

O programa solicita ao usuário que insira um número inteiro usando `printf()` e `scanf()` para ler a entrada do usuário e armazenar o valor na variável `número`.

Em seguida, ele utiliza um desvio condicional composto para verificar as seguintes condições:

Se o número for maior que zero (`número > 0`), ele imprime a mensagem "O número é positivo."

Se o número for menor que zero (`número < 0`), ele imprime a mensagem "O número é negativo."

ALGORITMOS E TÉCNICAS DE PROGRAMAÇÃO

Caso contrário (se o número for igual a zero), ele imprime a mensagem "O número é zero."

O programa então encerra sua execução.

3. Verificação de Aprovação Escolar: esse exemplo vai ilustrar como usar um desvio condicional composto para tomar decisões com base em uma condição específica, neste caso, para determinar se um aluno pode ou não passar de ano com base em sua média de notas.

```

1  #include <stdio.h>
2
3  int main() {
4      float media;
5
6      // Sólicita ao usuário que insira a média do aluno
7      printf("Digite a média do aluno: ");
8      scanf("%f", &media);
9
10     // Verifica se o aluno pode passar de ano
11     if (media >= 6.0) {
12         printf("O aluno foi aprovado e passará de ano!\n");
13     } else {
14         printf("O aluno foi reprovado e precisará fazer recuperação.\n");
15     }
16
17     return 0;
18 }
```

Figura 6 | Exemplo de Verificação de Aprovação Escolar em C.

Neste exemplo:

Solicitamos ao usuário que insira a média do aluno usando `printf()` e `scanf()` para ler a entrada e armazená-la na variável `media`.

Utilizamos um desvio condicional composto para verificar a seguinte condição:

Se a média do aluno for maior ou igual a 6.0 (`média >= 6.0`), isso significa que ele foi aprovado. Nesse caso, o programa imprime a mensagem "O aluno foi aprovado e passará de ano!"

Caso contrário (se a média for menor que 6.0), o aluno foi reprovado e precisará fazer recuperação. Nesse caso, o programa imprime a mensagem "O aluno foi reprovado e precisará fazer recuperação."

ALGORITMOS E TÉCNICAS DE PROGRAMAÇÃO

O programa então encerra sua execução.

Por fim, de acordo com os exemplos demonstrados podemos perceber como o desvio condicional composto "se/senão/fim-se" é utilizado na programação para tomar decisões com base em múltiplas possibilidades, proporcionando uma maior flexibilidade e controle no desenvolvimento de algoritmos e sistemas.

Siga em Frente...

Desvio condicional encadeado – testes condicionais encadeados

O desvio condicional encadeado, também conhecido como testes condicionais encadeados permitem a execução de múltiplos testes condicionais em sequência, em que cada teste depende do resultado do anterior, proporcionando uma abordagem mais flexível e adaptável para lidar com diversas situações.

A aplicação do desvio condicional encadeado é vasta e abrange desde a simples validação de entrada de dados até a implementação de lógicas de negócios mais elaboradas em sistemas de software.

Ao utilizar testes condicionais encadeados, os programadores têm a capacidade de criar algoritmos que respondam de forma dinâmica e eficiente a diferentes cenários, adaptando o comportamento do programa conforme necessário.

Esse método é de grande importância para a construção de sistemas completos e de qualidade, pois permite a implementação de fluxos de controle mais precisos e a execução de ações específicas com base em uma série de condições.

Compreender e saber como aplicar esse tipo de desvio fundamental para qualquer programador que deseje criar soluções tecnológicas de qualidade, capazes de lidar com diversas situações em que possa colocá-lo em prática.

A seguir, veremos três exemplos em que esse tipo de desvio pode ser aplicado para que você comprehenda melhor como ele funciona na prática.

1. Classificação de Triângulos: no exemplo demonstrado na Figura 7, o programa solicita ao usuário que insira os comprimentos dos lados de um triângulo e verifica qual tipo de triângulo é formado com base nesses comprimentos (equilátero, isósceles ou escaleno).

ALGORITMOS E TÉCNICAS DE PROGRAMAÇÃO

```
1 #include <stdio.h>
2
3 int main() {
4     int lado1, lado2, lado3;
5
6     printf("Digite o comprimento do lado 1: ");
7     scanf("%d", &lado1);
8     printf("Digite o comprimento do lado 2: ");
9     scanf("%d", &lado2);
10    printf("Digite o comprimento do lado 3: ");
11    scanf("%d", &lado3);
12
13    if (lado1 == lado2 && lado2 == lado3) {
14        printf("Triângulo equilátero.\n");
15    } else if (lado1 == lado2 || lado1 == lado3 || lado2 == lado3) {
16        printf("Triângulo isósceles.\n");
17    } else {
18        printf("Triângulo escaleno.\n");
19    }
20
21    return 0;
22 }
```

Figura 7 | Exemplo de Classificação de Triângulos em C.

2. Verificação de faixa etária: neste exemplo, o programa solicita ao usuário que insira sua idade e determina em qual faixa etária ele se encaixa (criança, adolescente, adulto ou idoso). Observe na Figura 8:

ALGORITMOS E TÉCNICAS DE PROGRAMAÇÃO

```
1 #include <stdio.h>
2
3 int main() {
4     int idade;
5
6     printf("Digite sua idade: ");
7     scanf("%d", &idade);
8
9     if (idade < 13) {
10         printf("Criança.\n");
11     } else if (idade >= 13 && idade < 18) {
12         printf("Adolescente.\n");
13     } else if (idade >= 18 && idade < 60) {
14         printf("Adulto.\n");
15     } else {
16         printf("Idoso.\n");
17     }
18
19     return 0;
20 }
```

Figura 8 | Exemplo de verificação de faixa etária em C.

3. Verificação de horário do dia: neste exemplo, o programa verifica o horário atual do dia e determina se é manhã, tarde, noite ou madrugada. Observe na Figura 9:

ALGORITMOS E TÉCNICAS DE PROGRAMAÇÃO

```
1  ✓ #include <stdio.h>
2    #include <time.h>
3
4  ✓ int main() {
5      time_t agora = time(NULL);
6      struct tm *local = localtime(&agora);
7      int hora = local->tm_hour;
8
9      ✓ if (hora >= 6 && hora < 12) {
10         printf("Bom dia!\n");
11     ✓ } else if (hora >= 12 && hora < 18) {
12         printf("Boa tarde!\n");
13     ✓ } else if (hora >= 18 && hora < 24) {
14         printf("Boa noite!\n");
15     ✓ } else {
16         printf("Boa madrugada!\n");
17     }
18
19     return 0;
20 }
```

Figura 9 | Exemplo de verificação de horário do dia em C.

Vamos Exercitar?

Descrição da situação-problema:

ALGORITMOS E TÉCNICAS DE PROGRAMAÇÃO

Você faz parte de uma equipe de programadores responsáveis por desenvolver um software para controle da nave espacial em uma missão de pesquisa na Lua. Durante a programação, surge a necessidade de implementar desvios condicionais para lidar com diferentes situações durante a viagem.

Enunciado:

Durante a viagem à Lua, a nave espacial pode encontrar algumas situações críticas que exigem tomada de decisões automáticas. Seu objetivo é implementar desvios condicionais utilizando o comando "if" na linguagem C para lidar com as seguintes situações:

- a. **Controle de combustível:** se o nível de combustível estiver abaixo de 20%, exiba uma mensagem de alerta.
- b. **Orientação da nave:** se a nave estiver fora da rota programada, corrija automaticamente a trajetória.
- c. **Condições climáticas:** se for identificada uma tempestade solar, ative o protocolo de proteção da nave.

Resolução da situação-problema:

Identificação dos dados:

1. Inicie o programa e leia o nível de combustível.
2. Aplique um desvio condicional "if" para verificar se o nível de combustível é inferior a 20%.
3. Se verdadeiro, exiba a mensagem: "Nível de combustível baixo. Alerta!"
4. Se falso, prossiga para o próximo passo.
5. Leia os dados de orientação da nave.
6. Utilize um desvio condicional "if" para verificar se a nave está fora da rota programada.
7. Se verdadeiro, acione o sistema de correção de trajetória.
8. Se falso, continue para o próximo passo.
9. Verifique as condições climáticas, identificando tempestades solares.
10. Aplique um desvio condicional "if" para ativar o protocolo de proteção da nave em caso de tempestade solar.
11. Se verdadeiro, exiba a mensagem: "Ativando protocolo de proteção contra tempestade solar."

Segue uma sugestão do algoritmo resolvido também na linguagem de programação C.

ALGORITMOS E TÉCNICAS DE PROGRAMAÇÃO

```
1  #include <stdio.h>
2  int main() {
3      // Passo 1: Leitura do nível de combustível
4      float nivelCombustivel;
5      printf("Informe o nível de combustível (em porcentagem): ");
6      scanf("%f", &nivelCombustivel);
7
8      // Passo 2: Desvio condicional para controle de combustível
9      if (nivelCombustivel < 20.0) {
10          printf("Nível de combustível baixo. Alerta!\n");
11      }
12
13     // Passo 3: Leitura dos dados de orientação da nave
14     int orientacaoNave;
15     printf("Informe a orientação da nave (1 para rota programada, 0 para fora da rota): ");
16     scanf("%d", &orientacaoNave);
17
18     // Passo 4: Desvio condicional para orientação da nave
19     if (orientacaoNave == 0) {
20         printf("Nave fora da rota programada. Corrigindo trajetória...\n");
21         // Lógica de correção de trajetória aqui
22     }
23
24     // Passo 5: Verificação de condições climáticas
25     int tempestadeSolar;
26     printf("Identificar tempestade solar (1 para sim, 0 para não): ");
27     scanf("%d", &tempestadeSolar);
28
29     // Passo 6: Desvio condicional para condições climáticas
30     if (tempestadeSolar == 1) {
31         printf("Ativando protocolo de proteção contra tempestade solar...\n");
32         // Lógica de proteção contra tempestade solar aqui
33     }
34
35 }
```

Figura 10 | Algoritmo resolvido.

Este código implementa a lógica descrita na situação problema utilizando desvios condicionais (*if*) para lidar com diferentes situações durante a viagem à Lua. Vale ressaltar que os comentários indicam os passos correspondentes da solução.

Assim, a aplicação adequada de desvios condicionais em algoritmos proporciona uma abordagem robusta para lidar com variáveis dinâmicas, garantindo o sucesso e a segurança de missões espaciais e outras aplicações críticas.

Saiba mais

ALGORITMOS E TÉCNICAS DE PROGRAMAÇÃO

Para saber mais sobre desvios condicionais aprofundando um pouco mais no assunto, acesse em sua Biblioteca Virtual o livro [*Introdução a Algoritmos e Programação com Python: Uma Abordagem Dirigida por Testes*](#), Capítulo 5 - páginas 51 a 68.

Referências

CORMEN, T. **Algoritmos – Teoria e Prática**. 3.ed. Rio de Janeiro: LTC, 2022.

CORMEN, T. **Desmistificando Algoritmos**. Grupo GEN, 2013. E-book.

FERREIRA, Aurélio Buarque de Holanda. **Novo Aurélio Século XXI**: o dicionário da língua portuguesa. 3 ed. totalmente rev. e ampl. Rio de Janeiro: Nova Fronteira, 1999.

MANZANO, J.A. N G.; OLIVEIRA, J. F. de. **Estudo Dirigido de Algoritmos**. 15. ed. Editora Saraiva, 2012. E-book.

MENÉNDEZ, A. **Simplificando Algoritmos**. Grupo GEN, 2023. E-book.

PINHEIRO, F. de A. C. **Elementos de Programação em C**. Porto Alegre: Bookman, 2012.

SERPA, M. S.; RODRIGUES, T. N.; ALVES, I. C.; et al. **Análise de Algoritmos**. Grupo A, 2021. E-book.

WAZLAWICK, R. S. **Introdução a Algoritmos e Programação com Python: Uma Abordagem Dirigida por Testes**. Rio de Janeiro: LTC, 2018.

Aula 2

Estruturas de Decisão Condisional (Caso Escolha)

Estruturas de decisão condicional (Caso Escolha)

Este conteúdo é um vídeo!



Para assistir este conteúdo é necessário que você acesse o AVA pelo computador ou pelo aplicativo. Você pode baixar os vídeos direto no aplicativo para assistir mesmo sem conexão à internet.

Dica para você

ALGORITMOS E TÉCNICAS DE PROGRAMAÇÃO

Aproveite o acesso para baixar os slides do vídeo, isso pode deixar sua aprendizagem ainda mais completa.

Olá, estudante! Nesta videoaula, vamos explorar o conceito da estrutura "Caso Escolha", descobrindo como ela pode ser poderosa para direcionar o fluxo do seu código. Se você deseja fortalecer suas habilidades de programação e entender como tomar decisões dinâmicas em seus algoritmos, não perca essa oportunidade! Venha conosco desbravar o universo das decisões na linguagem C. Para que você compreenda os conteúdos da melhor forma possível, vamos trazer exemplos em que esses conteúdos podem ser aplicados na prática, tornando seu aprendizado mais significativo.

Ponto de Partida

Olá, estudante!

Seja bem-vindo à próxima etapa da nossa jornada no universo da programação! Estamos prestes a explorar um conceito presente na programação: o desvio condicional com a estrutura "Caso Escolha".

Essa ferramenta é como uma chave mestra que permite direcionar o caminho do nosso programa com base em diferentes condições. Veremos como essa estrutura pode ser uma aliada poderosa na tomada de decisões dinâmicas em seus algoritmos.

Nesta fase, vamos mergulhar mais fundo em como podemos criar ramificações em nossos algoritmos, adaptando-os a diferentes situações.

Então, prepare-se para aprofundar seus conhecimentos em desvios condicionais com a estrutura "Caso Escolha" e tornar seu aprendizado, uma experiência empolgante e recompensadora.

Diante disso, será apresentado o desenvolvimento de um algoritmo que tem o objetivo de classificar os alunos em dois grupos distintos com base em suas notas.

Vamos lá?

Vamos Começar!

Estrutura de seleção Caso – Teste simples

ALGORITMOS E TÉCNICAS DE PROGRAMAÇÃO

As estruturas de seleção definitivamente são fundamentais, além disso, são muito utilizadas no desenvolvimento de algoritmos. Poderíamos dizer sem medo de errar, que não existe nenhum software sem que haja uma ou várias instruções de seleção dentro do seu código-fonte.

Se tal fato, fosse uma verdade, teríamos softwares totalmente estáticos, que não fizessem absolutamente nada sem ficar capturando e armazenando informações totalmente sem propósito.

Imagine uma situação em que uma professora faz um desafio a seus alunos para escreverem a melhor redação sobre o meio ambiente. Todos os alunos sem exceção se empenham em se concentrar, pretendendo realmente escrever um texto que impressione a professora.

E então, todos se apresentam na frente dos colegas e não existe ganhador, ou a pessoa que escreveu a melhor redação não recebe ao menos um "parabéns", ou ainda, ninguém fica sabendo qual foi a melhor redação. Pois bem, isso é mesmo frustrante.

Para que haja alguma dessas ações, as estruturas de seleção são fundamentais, pois mediante a elas irão existir condições, por exemplo, o aluno com o melhor desempenho, ou maior nota vencerá o desafio. Portanto, existe uma condição em que o vencedor vai receber a recompensa. E todos ficam ciente deste critério.

Em um outro exemplo, podemos pensar em um aluno de uma faculdade em que o pai promete a ele uma viagem internacional se ele for aprovado em todas as disciplinas com média 8.5 ou mais, além de ter uma frequência em todas as disciplinas de pelo menos 75%. Como construiríamos esse algoritmo senão com condicionais? Que são essas as estruturas de seleção.

Portanto, fica evidente que essas estruturas são realmente presentes em desenvolvimento de sistemas.

Estrutura de seleção Caso

Vamos iniciar nosso estudo nessa aula, abordando a instrução de seleção Caso, ou na linguagem C "**switch case**".

A estrutura de seleção Caso, também conhecida como *switch case* em algumas linguagens de programação, é uma ferramenta fundamental para controlar o fluxo de um programa. Ela permite que diferentes ações sejam executadas com base no valor de uma expressão.

A estrutura de seleção "Caso" é utilizada quando se deseja realizar diferentes ações com base no valor de uma variável. Ela é uma alternativa ao uso de múltiplas instruções "*if-else*" encadeadas, tornando o código mais legível e fácil de dar manutenção.

Na linguagem C, a estrutura switch case é utilizada da seguinte forma:

ALGORITMOS E TÉCNICAS DE PROGRAMAÇÃO

```
switch (expressao) {  
    case valor1:  
        // código a ser executado se expressao for igual a valor1  
        break;  
}
```

A expressão dentro do *switch* é avaliada e comparada com o valor do *case*. Se houver uma correspondência, o código dentro desse *case* é executado. O comando *break* é utilizado para sair do *switch* após a execução de um *case*. Ainda podemos acrescentar um outro bloco que será responsável por executar comandos caso o *case* não corresponda a expressão.

A Figura 1, a seguir, apresenta um exemplo da instrução na linguagem de programação C, a partir de um valor numérico solicitado pelo usuário. Caso ele escolha a opção "1", o código desviará para dentro do *case* e vai exibir a mensagem na tela: "Você escolheu a opção 1". Observe:

```
1  #include <stdio.h>  
2  
3  int main() {  
4      int escolha;  
5  
6      // Solicita ao usuário que faça uma escolha (1 ou 2)  
7      printf("Escolha 1 ou 2: ");  
8      scanf("%d", &escolha);  
9  
10     // Utiliza a instrução "Caso" para tratar a escolha do usuário  
11     switch (escolha) {  
12         case 1:  
13             printf("Você escolheu a opção 1.\n");  
14             break;  
15         // Não há "default" neste exemplo  
16     }  
17  
18     return 0;  
19 }  
20 }
```

Figura 1 | Exemplo da instrução “case” na Linguagem C.

ALGORITMOS E TÉCNICAS DE PROGRAMAÇÃO

Desta maneira, a exemplo do desvio condicional simples com o comando "if", a Figura 1 trata somente se o usuário escolher uma única opção. Para que ele trate as demais opções, temos que adicionar mais instruções case dentro do switch.

Estrutura e seleção caso com múltiplas opções

Neste momento, vamos trabalhar com a possibilidade de ter várias opções de execução para as condições. Quando temos uma, duas ou até três condições, podemos utilizar a instrução "if" normalmente, porém quando temos um número maior de condições o ideal é optar pelo uso do "switch case".

Um exemplo prático de aplicação da estrutura switch case em um contexto profissional pode ser encontrado em um sistema de vendas. Suponha que um sistema precise calcular o desconto a ser aplicado a um determinado produto com base no seu tipo. Poderíamos utilizar a estrutura switch case da seguinte forma:

```
```c
int tipoProduto = 2;
float preco = 100.0;
float desconto;
switch (tipoProduto) {
 case 1:
 desconto = preco * 0.1; // 10% de desconto para produtos do tipo 1
 break;
 case 2:
 desconto = preco * 0.2; // 20% de desconto para produtos do tipo 2
 break;
 case 3:
 desconto = preco * 0.3; // 30% de desconto para produtos do tipo 3
 break;
 case 4:
 desconto = preco * 0.4; // 40% de desconto para produtos do tipo 4
 break;
 default:
 desconto = 0; // nenhum desconto para outros tipos de produtos
}
float precoComDesconto = preco - desconto;
printf("O preço com desconto é: %.2f", precoComDesconto);
```

```

Neste exemplo, a variável tipoProduto é utilizada como expressão no *switch case*. Dependendo do valor de tipoProduto, um desconto diferente é calculado e aplicado ao preço do produto.

ALGORITMOS E TÉCNICAS DE PROGRAMAÇÃO

Outro exemplo cotidiano de aplicação da estrutura *switch case* pode ser encontrado em um programa para calcular o dia da semana com base no número correspondente ao dia. Por exemplo:

```
```c
int numeroDia = 3;
switch (numeroDia) {
 case 1:
 printf("Domingo");
 break;
 case 2:
 printf("Segunda-feira");
 break;
 // outros cases para os demais dias da semana
 default:
 printf("Dia inválido");
}
````
```

Neste exemplo, o valor (3) é adicionado na variável "numeroDia", portanto, a mensagem exibida em tela será: "**Dia inválido**", pois no código ainda não consta o tratamento para os outros dias da semana.

Para entender o que qualquer código está fazendo, temos que realizar o teste de mesa, que nada mais é do que simular a execução passo a passo do código fonte. Com o tempo e prática você conseguirá fazer isso de cabeça. Já imaginou que legal?

Mais um exemplo de execução do "*switch case*" com várias opções você consegue conferir na Figura 2. Observe:

ALGORITMOS E TÉCNICAS DE PROGRAMAÇÃO

```
1 #include <stdio.h>
2
3 int main() {
4     int opcao;
5
6     // Apresenta opções de filmes disponíveis
7     printf("Escolha um filme:\n");
8     printf("1 - Ação\n");
9     printf("2 - Comédia\n");
10    printf("3 - Drama\n");
11    printf("4 - Animação\n");
12    printf("5 - Suspense\n");
13
14    // Sólicita ao usuário que faça uma escolha (1 a 5)
15    printf("Digite o número do filme desejado: ");
16    scanf("%d", &opcao);
17
18    // Utiliza a instrução "Caso" para tratar a escolha do usuário
19    switch (opcao) {
20        case 1:
21            printf("Você escolheu o filme de Ação.\n");
22            break;
23        case 2:
24            printf("Você escolheu o filme de Comédia.\n");
25            break;
26        case 3:
27            printf("Você escolheu o filme de Drama.\n");
28            break;
29        case 4:
30            printf("Você escolheu o filme de Animação.\n");
31            break;
32        case 5:
33            printf("Você escolheu o filme de Suspense.\n");
34            break;
35        default:
36            printf("Opção inválida. Por favor, escolha um número de 1 a 5.\n");
37    }
38
39    return 0;
40 }
```

Figura 2 | Exemplo da instrução "case" com múltiplas opções.

No exemplo da Figura 2, é exibido um menu com informações do gênero de filmes para que o usuário faça sua escolha. Após lida a opção o "switch case" é executado e vai direto para a

ALGORITMOS E TÉCNICAS DE PROGRAMAÇÃO

instrução em que a variável "opcao", de acordo com seu valor, executará.

Siga em Frente...

Estudo de caso aplicando a estrutura de seleção caso

Abordaremos, agora, um estudo de caso para demonstrar na prática como essa estrutura trabalha, bem como, suas características e técnicas utilizadas no desenvolvimento de algoritmos.

Vamos começar pela contextualização ou a ideia inicial o estudo de caso.

Contextualização: Em um fast food, é comum lidar com uma variedade de pedidos que incluem lanches, combos, bebidas e condimentos extras. Para tornar o processo mais eficiente, criaremos um sistema simples utilizando a linguagem de programação C, que utiliza a estrutura de seleção *switch case*. O programa solicitará um código e senha do atendente no início para garantir acesso autorizado.

Passo 1: Solicitar código e senha do atendente:

- Se o código e a senha forem corretos, avance para o próximo passo.
- Caso contrário, exiba uma mensagem de acesso não autorizado e encerre o programa.

Passo 2: Apresentar opções de pedido:

- Lanche (Opção 1).
- Combo (Opção 2).
- Bebida (Opção 3).
- Condimentos extras (Opção 4).

Passo 3: Utilizar a estrutura "*switch case*" para lidar com as opções escolhidas pelo atendente:

- Para cada opção, solicitar os detalhes do pedido e exibir uma confirmação.

Passo 4: Calcular o valor total do pedido com base nas escolhas do atendente:

- Codificação na Linguagem de Programação C.

Vamos analisar agora a codificação para atender o estudo de caso. No primeiro momento é realizada as solicitações do código e senha do atendente. Cada informação é gravada em uma variável específica com o auxílio do comando *scanf()*, como pode ser observado nas linhas 7 e 9 da Figura 3.

ALGORITMOS E TÉCNICAS DE PROGRAMAÇÃO

```

4      // Passo 1: Autenticação do atendente
5      int codigo, senha;
6      printf("Digite o código do atendente: ");
7      scanf("%d", &codigo);
8
9      printf("Digite a senha: ");
10     scanf("%d", &senha);
11

```

Figura 3 | Criação e inicialização das variáveis.

Na sequência, o algoritmo verifica se o código e senha informado são condizentes ao que está na condição "*if*". Note que já estamos utilizando uma seleção, porém neste momento é o comando "*if*". Observe a Figura 4.

```

12      if (codigo == 123 && senha == 456) {
13          // Passo 2: Apresentar opções de pedido
14          int opcao;
15          printf("\nEscolha uma opção:\n");
16          printf("1 - Lanche\n");
17          printf("2 - Combo\n");
18          printf("3 - Bebida\n");
19          printf("4 - Condimentos extras\n");
20
21          // Passo 3: Utilizar a estrutura switch case
22          printf("\nDigite o número da opção desejada: ");
23          scanf("%d", &opcao);
24

```

Figura 4 | Teste do código e senha.

A partir da linha 14, se o código passar pela validação da instrução "*if*", ou seja, se o código for "123" e a senha for "456", é exibido o menu com as opções disponível ao usuário. Cabe a ele escolher uma das opções e pressionar a tecla ENTER.

Diante disso, a instrução em questão: "*switch case*" entra em ação e vai testar a opção do usuário por meio do conteúdo da variável "opcao". Isso ocorre a partir da linha 25. Observe a Figura 5.

ALGORITMOS E TÉCNICAS DE PROGRAMAÇÃO

```
25     switch (opcao) {
26         case 1:
27             printf("\nVocê escolheu Lanche. Por favor, informe os detalhes do pedido.\n");
28             // Código para lidar com pedidos de lanche
29             break;
30         case 2:
31             printf("\nVocê escolheu Combo. Por favor, informe os detalhes do pedido.\n");
32             // Código para lidar com pedidos de combo
33             break;
34         case 3:
35             printf("\nVocê escolheu Bebida. Por favor, informe os detalhes do pedido.\n");
36             // Código para lidar com pedidos de bebida
37             break;
38         case 4:
39             printf("\nVocê escolheu Condimentos extras. Por favor, informe os detalhes do pedido.\n");
40             // Código para lidar com pedidos de condimentos extras
41             break;
42         default:
43             printf("\nOpção inválida.\n");
44     }
45
46     // Passo 4: Cálculo do valor total
47     // Código para calcular o valor total do pedido
48
49 } else {
50     printf("\nAcesso não autorizado. Encerrando o programa.\n");
51 }
```

Figura 5 | Instrução "switch case".

Observe que, ainda no final do código, na linha 50, o algoritmo exibe ao usuário uma mensagem dizendo que o acesso não foi autorizado, caso a validação do código e senha solicitados pelo atendente não sejam coerentes.

O uso da estrutura "*switch case*" proporciona uma abordagem organizada e eficiente para lidar com múltiplas opções em um programa. Neste estudo de caso, observamos como ela pode ser aplicada em um sistema de pedidos de fast food, tornando o código mais legível e fácil de manter.

A flexibilidade do *switch case* é valiosa em situações em que diversas escolhas precisam ser consideradas, destacando sua importância no desenvolvimento de software estruturado.

Vamos Exercitar?

Descrição da situação-problema:

Em uma escola, após a aplicação de uma prova, é necessário classificar os alunos em dois grupos distintos com base em suas notas. O objetivo é identificar quais alunos obtiveram notas iguais ou superiores a 70 e quais ficaram abaixo desse patamar.

ALGORITMOS E TÉCNICAS DE PROGRAMAÇÃO

Desafio:

Para resolver o problema, você precisa criar um algoritmo na linguagem de programação C utilizando a estrutura switch case para resolver o problema.

Resolução da situação-problema:

Entrada de Dados:

- Solicite ao usuário que insira a nota do aluno.
- Armazene essa nota em uma variável.

```
#include <stdio.h>
int main() {
    float nota;
    printf("Digite a nota do aluno: ");
    scanf("%f", &nota);
```

Estrutura de Seleção "Caso Escolha"

- Utilize a estrutura "Caso Escolha" para classificar o aluno em um dos grupos.
- Se a nota for igual ou superior a 70, o aluno pertence ao Grupo 1; caso contrário, pertence ao Grupo 2.

```
switch(nota) {
    case nota >= 70:
        printf("Aluno pertence ao Grupo 1.\n");
        break;
    default:
        printf("Aluno pertence ao Grupo 2.\n");
        break;
}
```

Saída de Resultado

- Exiba uma mensagem indicando a qual grupo o aluno pertence.

Teste e Ajustes

ALGORITMOS E TÉCNICAS DE PROGRAMAÇÃO

- Teste o algoritmo com diferentes notas para garantir seu correto funcionamento.
- Faça ajustes conforme necessário.

Este algoritmo simples permite classificar os alunos em dois grupos com base em suas notas, utilizando a estrutura de seleção "Caso Escolha". Essa prática ajuda a compreender como aplicar condicionais para tomar decisões específicas em algoritmos.

Experimente modificar e expandir o algoritmo para incluir mais casos ou adapte-o conforme a necessidade do cenário apresentado. Isso contribuirá para fortalecer seu entendimento sobre estruturas de seleção.

Saiba mais

Para saber mais sobre desvios condicionais aprofundando um pouco mais no assunto, especificamente sobre estruturas condicionais - (Comando *switch case*) acesse em sua Biblioteca Virtual o livro [*Elementos de Programação em C*](#), Capítulo 7, seção 2 - páginas 153 a 161.

Referências

CORMEN, T. **Algoritmos – Teoria e Prática**. 3.ed. Rio de Janeiro: LTC, 2022.

CORMEN, T. **Desmistificando Algoritmos**. Grupo GEN, 2013. E-book.

FERREIRA, Aurélio Buarque de Holanda. **Novo Aurélio Século XXI**: o dicionário da língua portuguesa. 3 ed. totalmente rev. e ampl. Rio de Janeiro: Nova Fronteira, 1999.

MANZANO, J.A. N G.; OLIVEIRA, J. F. de. **Estudo Dirigido de Algoritmos**. 15. ed. Editora Saraiva, 2012. E-book.

MENÉNDEZ, A. **Simplificando Algoritmos**. Grupo GEN, 2023. E-book.

PINHEIRO, F. de A. C. **Elementos de Programação em C**. Porto Alegre: Bookman, 2012.

SERPA, M. S.; RODRIGUES, T. N.; ALVES, I. C.; et al. **Análise de Algoritmos**. Grupo A, 2021. E-book.

WAZLAWICK, R. S. **Introdução a Algoritmos e Programação com Python: Uma Abordagem Dirigida por Testes**. Rio de Janeiro: LTC, 2018.

ALGORITMOS E TÉCNICAS DE PROGRAMAÇÃO

Aula 3

Estruturas de Repetição Condicional

Estruturas de repetição condicional



Este conteúdo é um vídeo!

Para assistir este conteúdo é necessário que você acesse o AVA pelo computador ou pelo aplicativo. Você pode baixar os vídeos direto no aplicativo para assistir mesmo sem conexão à internet.

Dica para você

Aproveite o acesso para baixar os slides do vídeo, isso pode deixar sua aprendizagem ainda mais completa.

Olá, estudante!

Nesta videoaula, abordaremos assuntos relativos à algoritmos e técnicas de programação como: Estruturas de repetição condicional - Enquanto Faça, Estruturas de repetição condicional - Faça Enquanto e Estruturas de repetição condicional - Repita Até.

Para que você comprehenda os conteúdos da melhor forma possível, vamos trazer exemplos em que esses conteúdos podem ser aplicados na prática, tornando seu aprendizado mais significativo.

Os conhecimentos adquiridos aqui não apenas expandem a sua compreensão do assunto, mas também fortalecem a sua capacidade de pensamento crítico e resolução de problemas. Não perca essa oportunidade de aprimorar suas habilidades em programação.

Ponto de Partida

Olá, estudante! Nesta aula, exploraremos os conteúdos referentes a algoritmos e técnicas de programação e teremos como foco o estudo nos diferentes tipos de "laços de repetições" ou simplesmente *loopings*, como também são conhecidos.

Aprender os comandos de laços de repetição na linguagem de programação C é essencial para desenvolver códigos mais eficientes, econômicos e de fácil manutenção.

Veremos as razões pelas quais esse conhecimento é fundamental, por exemplo: a eficiência computacional, é alcançada, utilizando laços de repetição. Podemos executar um conjunto de

ALGORITMOS E TÉCNICAS DE PROGRAMAÇÃO

instruções várias vezes sem a necessidade de reescrever o código. Isso economiza recursos computacionais e melhora a eficiência do programa.

Outro fator importante é a economia de tempo e esforço. Ao automatizar processos repetitivos, economizamos tempo e esforço no desenvolvimento de software. Isso permite que os programadores se concentrem em aspectos mais complexos e inovadores de seus projetos.

Sem mencionar, também, a resolução de problemas complexos. Muitos problemas do mundo real exigem a aplicação de algoritmos que envolvem a repetição controlada. A habilidade de implementar laços de repetição é vital para resolver desafios mais complexos.

Diante disso, será apresentado um exemplo clássico utilizando laços de repetições. O desenvolvimento de um algoritmo que tem o objetivo de fazer o **cálculo da tabuada** de um número qualquer escolhido na execução do programa.

Convidamos você a iniciar a construção de uma base sólida para sua jornada no universo da programação. Preparado para essa imersão?

Vamos Começar!

Laços de Repetições

Aprofundando um pouco mais em nossa disciplina, ao passo em que vamos descobrindo e estudando novas instruções e estruturas de programação, vamos também fixando melhor todos os assuntos já abordados.

Assim é exatamente quando se começa a estudar algoritmos. Primeiro, devemos estudar os fundamentos, características e aspectos como comandos e estruturas. Se pudéssemos escrever uma linha do tempo de como é a ordem dos conteúdos em que devem ser estudados e compreendidos, poderíamos ter algo parecido com a Figura 1. Observe:

ALGORITMOS E TÉCNICAS DE PROGRAMAÇÃO



Figura 1 | Linha do tempo de estudo de algoritmos.

ALGORITMOS E TÉCNICAS DE PROGRAMAÇÃO

Nela, encontra-se um manual que tem que ser seguido à risca, porém algumas dicas que se forem seguidas, certamente você encontrará êxito.

Neste aprendizado constante, deparamos-nos com as estruturas de repetição ou laços de repetições, estruturas capazes de fazer com que uma única linha de código seja executada mais do que uma única vez. Isso não é mágico? No primeiro momento, podemos até achar que isso não seria possível, mas é.

Podemos afirmar, ainda, que é muito utilizado em todos os sistemas. Podemos citar aqui inúmeros exemplos:

- **Sistemas de controle de estoque:** laços de repetição podem ser utilizados para processar grandes volumes de dados em inventários, atualizando quantidades, verificando prazos de validade e gerando relatórios.
- **Sistemas bancário on-line:** em sistemas bancários, laços de repetição são empregados para realizar operações em lote, como o processamento de transações automáticas, atualização de saldos e geração de extratos.
- **Sistema de gerenciamento de pedidos em um restaurante on-line:** na área de alimentos, sistemas que lidam com pedidos online podem utilizar laços de repetição para processar e atualizar o status dos pedidos, notificar clientes sobre atualizações e gerar relatórios de desempenho.
- **Sistema de processamento de imagens médicas:** em sistemas médicos que processam imagens, laços de repetição podem ser empregados para analisar pixels, identificar padrões e realizar operações complexas em grandes conjuntos de dados de imagens médicas.
- **Sistema de monitoramento de tráfego em tempo real:** sistemas de monitoramento de tráfego utilizam laços de repetição para processar continuamente dados de várias fontes, atualizar informações em tempo real, identificar padrões de tráfego e fornecer análises para otimização de rotas.
- **Jogos digitais:** em jogos digitais o *looping*, ou laços de repetições também são muito utilizados. Por exemplo para que um personagem se desloque na tela de um jogo, é necessário programar um comando para que ele dê um passo na direção escolhida. Para que ele ande uma grande distância nesta mesma direção, basta colocar este comando dentro de um comando de laço. É dessa maneira que se faz a mágica.

Realmente são muitos exemplos. Agora dentro da linguagem de programação C vamos estudar cada um dos comandos de laços de repetições existentes.

Enquanto Faça (*while*)

Os laços de repetições são estruturas fundamentais em qualquer linguagem de programação, incluindo a linguagem C. Eles permitem que um conjunto de instruções seja executado repetidamente enquanto uma condição específica for verdadeira. Existem três tipos principais de

ALGORITMOS E TÉCNICAS DE PROGRAMAÇÃO

Iaços de repetições em C: **while**, **for** e **do while**. Neste conteúdo, exploraremos cada um desses comandos, fornecendo exemplos e aplicações práticas.

O comando **while** é utilizado para repetir um bloco de código enquanto uma condição específica for verdadeira. A estrutura básica do comando **while** em C é a seguinte:

```
```c
while (condição) {
 // bloco de código a ser repetido
}
```

O comando **while** é especialmente útil quando o número de iterações não é conhecido antecipadamente. Por exemplo, imagine um programa que solicita ao usuário para digitar números até que ele decida parar. Nesse caso, o comando **while** pode ser utilizado da seguinte forma:

```
```c
#include <stdio.h>
int main() {
    int numero;
    while (1) {
        printf("Digite um número (ou 0 para sair): ");
        scanf("%d", &numero);
        if (numero == 0) {
            break;
        }
    }
    return 0;
}
````
```

Um exemplo prático do uso do comando **while** é em sistemas de controle de estoque, onde um conjunto de ações precisa ser repetido até que determinada condição seja atendida, como a atualização de quantidades de produtos em um sistema. Observe um exemplo desse sistema na Figura 2.

# ALGORITMOS E TÉCNICAS DE PROGRAMAÇÃO

```
1 #include <stdio.h>
2
3 int main() {
4 // Variáveis para simular o controle de estoque
5 int quantidadeAtual = 100;
6 int quantidadeMinima = 50;
7
8 // Mensagem inicial
9 printf("Controle de Estoque\n");
10
11 // Loop enquanto a quantidade atual for maior que a quantidade mínima
12 while (quantidadeAtual > quantidadeMinima) {
13 // Exibição da quantidade atual e solicitação de atualização
14 printf("Quantidade Atual: %d\n", quantidadeAtual);
15 printf("Digite a quantidade a ser atualizada (ou 0 para sair): ");
16
17 // Leitura da quantidade a ser atualizada
18 int atualizacao;
19 scanf("%d", &atualizacao);
20
21 // Verificação se a atualização é válida
22 if (atualizacao < 0) {
23 printf("Quantidade inválida. Tente novamente.\n");
24 } else if (atualizacao == 0) {
25 // Saída do loop se a atualização for 0 (usuário escolheu sair)
26 printf("Saindo do Controle de Estoque.\n");
27 break;
28 } else {
29 // Atualização da quantidade
30 quantidadeAtual += atualizacao;
31 printf("Estoque atualizado com sucesso!\n");
32 }
33 }
34
35 // Mensagem de encerramento
36 printf("Fim do Controle de Estoque.\n");
37
38 return 0;
39}
```

Figura 2 | Comando while.

# ALGORITMOS E TÉCNICAS DE PROGRAMAÇÃO

Siga em Frente...

## Para Faça (*for*)

O comando **for** é outra estrutura de repetição em C, porém é mais adequado quando o número de iterações é conhecido antecipadamente. A estrutura básica do comando **for** em C é a seguinte:

```
```c
for (inicialização; condição; incremento) {
    // bloco de código a ser repetido
}
````
```

O comando **for** é amplamente utilizado em situações em que é necessário iterar sobre uma sequência de números ou elementos. Por exemplo, para imprimir os números de 1 a 10, podemos utilizar o comando **for** da seguinte forma:

```
```c
#include <stdio.h>
int main() {
    for (int i = 1; i <= 10; i++) {
        printf("%d\n", i);
    }
    return 0;
}
````
```

Um exemplo prático do uso do comando **for** é em algoritmos de busca e ordenação, onde é necessário percorrer um conjunto de dados de forma sequencial. Observe um exemplo desse sistema na Figura 3.

# ALGORITMOS E TÉCNICAS DE PROGRAMAÇÃO

```
1 #include <stdio.h>
2
3 int main() {
4 // Tamanho do conjunto de dados
5 int tamanho = 5;
6
7 // Conjunto de dados (pode ser substituído por um vetor real)
8 int dados[] = {10, 4, 8, 2, 6};
9
10 // Mensagem inicial
11 printf("Conjunto de Dados: ");
12
13 // Exibição do conjunto de dados
14 for (int i = 0; i < tamanho; i++) {
15 printf("%d ", dados[i]);
16 }
17
18 // Ordenação do conjunto de dados (método simples de ordenação - bubble sort)
19 for (int i = 0; i < tamanho - 1; i++) {
20 for (int j = 0; j < tamanho - i - 1; j++) {
21 // Troca dos elementos se estiverem fora de ordem
22 if (dados[j] > dados[j + 1]) {
23 int temp = dados[j];
24 dados[j] = dados[j + 1];
25 dados[j + 1] = temp;
26 }
27 }
28 }
29
30 // Mensagem após ordenação
31 printf("\nConjunto de Dados Ordenado: ");
32
33 // Exibição do conjunto de dados ordenado
34 for (int i = 0; i < tamanho; i++) {
35 printf("%d ", dados[i]);
36 }
37
38 // Mensagem de encerramento
39 printf("\nOrdenação concluída.\n");
40
41 return 0;
42 }
```

Figura 3 | Comando for.

## Repita Até (*do while*)

# ALGORITMOS E TÉCNICAS DE PROGRAMAÇÃO

O comando **do while** é semelhante ao comando **while**, porém garante que o bloco de código seja executado pelo menos uma vez, mesmo se a condição inicial for falsa. A estrutura básica do comando **do while** em C é a seguinte:

```
```c
do {
    // bloco de código a ser repetido
} while (condição);
````
```

O comando **do while** é útil em situações em que é necessário garantir que um bloco de código seja executado pelo menos uma vez, independentemente da condição. Por exemplo, em um jogo em que o jogador deve fazer pelo menos uma jogada antes de decidir se quer continuar ou sair.

```
```c
#include <stdio.h>

int main() {
    int escolha;
    do {
        printf("1 - Jogar novamente\n2 - Sair\nEscolha: ");
        scanf("%d", &escolha);
    } while (escolha != 2);
    return 0;
}
````
```

Um exemplo prático do uso do comando **do while** é em sistemas de menu interativos, onde o usuário deve fazer pelo menos uma seleção antes de sair do menu. Observe um exemplo desse sistema na Figura 4:

# ALGORITMOS E TÉCNICAS DE PROGRAMAÇÃO

```
1 #include <stdio.h>
2
3 int main() {
4 // Variável para armazenar a escolha do usuário
5 int escolha;
6
7 // Loop do menu
8 do {
9 // Exibição do menu
10 printf("\n*** Menu Interativo ***\n");
11 printf("1. Opção 1\n");
12 printf("2. Opção 2\n");
13 printf("3. Opção 3\n");
14 printf("0. Sair\n");
15
16 // Solicitação da escolha do usuário
17 printf("Escolha uma opção: ");
18 scanf("%d", &escolha);
19
20 // Verificação da escolha e execução da ação correspondente
21 switch (escolha) {
22 case 1:
23 printf("Você escolheu a Opção 1.\n");
24 // Lógica da Opção 1
25 break;
26 case 2:
27 printf("Você escolheu a Opção 2.\n");
28 // Lógica da Opção 2
29 break;
30 case 3:
31 printf("Você escolheu a Opção 3.\n");
32 // Lógica da Opção 3
33 break;
34 case 0:
35 printf("Saindo do Menu. Até logo!\n");
36 break;
37 default:
38 printf("Opção inválida. Tente novamente.\n");
39 }
40 } while (escolha != 0); // O loop continua até o usuário escolher sair (opção 0)
41
42 return 0;
43 }
44 }
```

Figura 4 | do while.

Os laços de repetições são essenciais para a construção de algoritmos eficientes e poderosos em linguagem C. Ao dominar o uso dos comandos *while*, *for* e *do while*, os programadores

# ALGORITMOS E TÉCNICAS DE PROGRAMAÇÃO

podem criar soluções mais elegantes e funcionais para uma variedade de problemas do mundo real.

É importante praticar e experimentar com essas estruturas para compreender completamente seu potencial e aplicabilidade.

## Vamos Exercitar?

### Descrição da situação-problema:

Imagine que você está desenvolvendo um programa educacional para crianças que ajuda a praticar a tabuada de multiplicação. O programa deve solicitar à criança um número específico e, em seguida, exibir a tabuada de multiplicação desse número de 1 a 10. Se a resposta da criança estiver correta, ela avança para o próximo número; caso contrário, recebe uma nova chance.

### Enunciado:

O seu trabalho será desenvolver esse algoritmo na linguagem de programação C, se atentando com as especificações e regras para a construção dele.

### Resolução da situação-problema:

#### 1. Solicitar Número

- **Entrada:** Solicitar à criança que digite um número inteiro para ver a tabuada desse número.

#### 2. Exibir Tabuada

- **Processamento:** Utilizar um laço de repetição "**for**" para calcular e exibir a tabuada desse número de 1 a 10.

# ALGORITMOS E TÉCNICAS DE PROGRAMAÇÃO

```

1 #include <stdio.h>
2
3 int main() {
4
5 int numero;
6 printf("Digite um número inteiro para a tabuada: ");
7 scanf("%d", &numero);
8
9 for (int i = 1; i <= 10; i++) {
10 printf("%d x %d = %d\n", numero, i, numero * i);
11 }
12
13 return 0;
14 }
```

Figura 5 | Laço for.

### 3. Verificar Resposta

- Entrada:** Solicitar à criança que responda qual é o resultado da multiplicação.
- Processamento:** Comparar a resposta da criança com o resultado correto.

```

13 int resposta;
14 printf("Qual é o resultado da multiplicação de %d por 5? ", numero);
15 scanf("%d", &resposta);
16
17 if (resposta == numero * 5) {
18 printf("Resposta Correta! Avance para o próximo número.\n");
19 } else {
20 printf("Resposta Incorreta. Tente novamente.\n");
21 }
```

Figura 6 | Condicional if-else.

### 4. Repetir ou Encerrar

- Processamento:** Verificar se a criança deseja praticar outra tabuada ou encerrar o programa.
- Abaixo o código completo da situação problema:

# ALGORITMOS E TÉCNICAS DE PROGRAMAÇÃO

```
1 #include <stdio.h>
2
3 int main() {
4 char opcao;
5
6 do {
7 int numero, resposta;
8
9 // Solicitar Número
10 printf("Digite um número inteiro para a tabuada: ");
11 scanf("%d", &numero);
12
13 // Exibir Tabuada
14 printf("Tabuada de %d:\n", numero);
15 for (int i = 1; i <= 10; i++) {
16 printf("%d x %d = %d\n", numero, i, numero * i);
17 }
18
19 // Verificar Resposta
20 printf("Qual é o resultado da multiplicação de %d por 5? ", numero);
21 scanf("%d", &resposta);
22
23 if (resposta == numero * 5) {
24 printf("Resposta Correta! Avance para o próximo número.\n");
25 } else {
26 printf("Resposta Incorreta. Tente novamente.\n");
27 }
28
29 // Repetir ou Encerrar
30 printf("Deseja praticar outra tabuada? (S/N) ");
31 scanf(" %c", &opcao);
32
33 } while (opcao == 'S' || opcao == 's');
34
35 printf("Programa encerrado. Obrigado por praticar!\n");
36
37 return 0;
38 }
```

Figura 7 | Laço do-while.

Essa situação problema incentiva o aprendizado lúdico da tabuada de multiplicação, proporcionando uma prática interativa. O programa oferece uma abordagem educacional

# ALGORITMOS E TÉCNICAS DE PROGRAMAÇÃO

envolvente e reforça os conceitos de laços de repetição e estrutura condicional na linguagem C.

## Saiba mais

Para saber mais sobre laços de repetições aprofundando um pouco mais no assunto, especificamente sobre estruturas de repetição acesse em sua Biblioteca Virtual o livro [Elementos de Programação em C](#), Capítulo 8 - páginas 171 a 190.

## Referências

CORMEN, T. **Algoritmos – Teoria e Prática**. 3.ed. Rio de Janeiro: LTC, 2022.

CORMEN, T. **Desmistificando Algoritmos**. Grupo GEN, 2013. E-book.

FERREIRA, Aurélio Buarque de Holanda. **Novo Aurélio Século XXI**: o dicionário da língua portuguesa. 3 ed. totalmente rev. e ampl. Rio de Janeiro: Nova Fronteira, 1999.

MANZANO, J.A. N G.; OLIVEIRA, J. F. de. **Estudo Dirigido de Algoritmos**. 15. ed. Editora Saraiva, 2012. E-book.

MENÉNDEZ, A. **Simplificando Algoritmos**. Grupo GEN, 2023. E-book.

PINHEIRO, F. de A. C. **Elementos de Programação em C**. Porto Alegre: Bookman, 2012.

SERPA, M. S.; RODRIGUES, T. N.; ALVES, Í. C.; *et al.* **Análise de Algoritmos**. Grupo A, 2021. E-book.

WAZLAWICK, R. S. **Introdução a Algoritmos e Programação com Python: Uma Abordagem Dirigida por Testes**. Rio de Janeiro: LTC, 2018.

## Aula 4

Estruturas de Repetição Determinísticas

## Videoaula

Este conteúdo é um vídeo!

# ALGORITMOS E TÉCNICAS DE PROGRAMAÇÃO



Para assistir este conteúdo é necessário que você acesse o AVA pelo computador ou pelo aplicativo. Você pode baixar os vídeos direto no aplicativo para assistir mesmo sem conexão à internet.

Dica para você

Aproveite o acesso para baixar os slides do vídeo, isso pode deixar sua aprendizagem ainda mais completa.

Olá, estudante! Nesta videoaula, veremos a fundo o histórico das estruturas de repetição e determinísticas, compreendendo sua evolução ao longo do tempo e como elas se tornaram fundamentais para a programação moderna.

Desde suas origens até as técnicas mais avançadas, você descobrirá como as estruturas de repetição determinísticas têm desempenhado um papel importante na automação de tarefas repetitivas e na otimização do desenvolvimento de software ao longo do tempo.

Ao longo da videoaula, faremos um interessante comparativo entre as estruturas de repetição determinísticas e as estruturas condicionais. Você aprenderá a identificar as diferenças essenciais entre esses dois conceitos, compreendendo quando e como aplicá-los de forma eficaz em seus projetos de programação.

Essa análise comparativa proporcionará uma compreensão mais profunda sobre como escolher a estrutura mais adequada para cada situação, garantindo um código mais claro e eficiente.

Você também terá a oportunidade de explorar as diversas aplicações das estruturas de repetição determinísticas em cenários do mundo real. Desde a manipulação de grandes volumes de dados até a automatização de processos complexos, essas estruturas desempenham um papel de grande relevância em uma variedade de domínios, incluindo desenvolvimento de software, ciência de dados, engenharia e muito mais.

Prepare-se para ampliar seu conhecimento e adquirir novas habilidades que impulsionarão os seus conhecimentos na programação.

## Ponto de Partida

Olá, estudante! Nesta aula, exploraremos conteúdos referentes a algoritmos e técnicas de programação e teremos como foco o estudo das estruturas de repetição determinísticas e o seu comparativo com as estruturas condicionais.

A compreensão desses conceitos e de seu histórico na programação se faz necessária para o futuro programador, pois por meio da descoberta de suas origens é que você poderá entender como as estruturas de repetição determinísticas têm desempenhado um papel importante na automação de tarefas repetitivas e na otimização do desenvolvimento de software ao longo do tempo.

Inicialmente, veremos como esses conceitos surgiram e se desenvolveram ao longo do tempo, moldando a forma como os algoritmos são criados e otimizados. Compreender o contexto

# ALGORITMOS E TÉCNICAS DE PROGRAMAÇÃO

histórico por trás dessas estruturas nos permitirá apreciar melhor sua importância e aplicabilidade na resolução de problemas computacionais.

Além disso, durante o percurso dessa aula, observaremos o comparativo entre as estruturas de repetição determinísticas e as estruturas condicionais, que será fundamental para aprofundar nossa compreensão sobre as diferenças e semelhanças entre esses dois conceitos, permitindo-nos discernir quando e como utilizar cada um deles de maneira inteligente.

Diante disso, será apresentado um exemplo contextualizando um site que vende produtos online no modelo de e-commerce. Será utilizado uma instrução de laços de repetições nesta abordagem.

Vamos lá?

## Vamos Começar!

### Histórico das estruturas de repetição determinísticas

Desde os primórdios da computação, a busca por eficiência e praticidade impulsionou o desenvolvimento de estruturas de repetição determinísticas, que desempenham um papel fundamental na criação de algoritmos.

Estas estruturas, ao permitirem a execução repetida de um bloco de código com base em uma condição predefinida, representam um marco na evolução da programação. Neste texto, exploraremos o histórico dessas estruturas, destacando sua importância e contribuição para o avanço da ciência da computação.

As estruturas de repetição determinísticas, também conhecidas como estruturas de repetição definidas ou estruturas de repetição controladas por contador, são elementos fundamentais da programação que permitem executar um conjunto de instruções repetidamente com base em uma condição específica.

Elas são caracterizadas por sua natureza previsível e controlada, em que o número de iterações é determinado antes da execução do loop e geralmente depende de uma variável de controle.

Em outras palavras, as estruturas de repetição determinísticas são aquelas em que o número de vezes que um bloco de código é executado é conhecido previamente e definido por um contador ou variável de controle. Elas são frequentemente utilizadas quando se sabe exatamente quantas vezes uma determinada ação precisa ser repetida, facilitando a criação de algoritmos mais eficientes e comprehensíveis.

# ALGORITMOS E TÉCNICAS DE PROGRAMAÇÃO

As estruturas de repetição determinísticas têm suas raízes na década de 1950, quando os primeiros computadores de grande porte foram desenvolvidos. Um dos primeiros exemplos dessas estruturas é o comando "FORTRAN DO", introduzido em 1957 no Fortran I, uma das primeiras linguagens de programação de alto nível.

Ao longo dos anos, as estruturas de repetição determinísticas evoluíram e foram incorporadas em diversas linguagens de programação modernas, como C, Python e Java. Sua importância reside na capacidade de automatizar tarefas repetitivas e lidar com volumes massivos de dados de maneira eficiente.

Por exemplo, em algoritmos de busca e ordenação, como o algoritmo de ordenação por seleção, as estruturas de repetição determinísticas são essenciais para percorrer listas de elementos e realizar as operações necessárias.

Outros exemplos comuns de estruturas de repetição determinísticas incluem o comando "for" em diversas linguagens de programação, como C, C++, Java e Python, e o comando "DO" em algumas versões de Fortran.

Essas estruturas permitem que os programadores criem loops com base em uma condição inicial, uma condição de continuação e uma expressão de incremento, controlando assim o fluxo de execução do programa de forma determinística.

Vamos contextualizar a parte histórica das estruturas de repetição determinísticas por meio de um exemplo comum em nosso dia a dia: "a prática de contar". Antes do desenvolvimento de algoritmos e linguagens de programação modernas, as pessoas contavam manualmente usando repetições determinísticas. Suponhamos que alguém queira contar o número de maçãs que colheu de uma determinada árvore.

A linguagem de programação C pode ser utilizada para criar um algoritmo que simula esse processo de contar usando uma estrutura de repetição **while**. O algoritmo poderia ser algo como:

# ALGORITMOS E TÉCNICAS DE PROGRAMAÇÃO

```
1 #include <stdio.h>
2
3 int main() {
4 // Variável para armazenar o número de maçãs
5 int numeroDeMacas = 0;
6
7 // Mensagem inicial
8 printf("Vamos contar o número de maçãs colhidas:\n");
9
10 // Loop enquanto o número de maçãs for menor que 10
11 while (numeroDeMacas < 10) {
12 // Incremento do número de maçãs a cada iteração
13 numeroDeMacas++;
14
15 // Mensagem indicando quantas maçãs foram contadas até agora
16 printf("Maçãs contadas: %d\n", numeroDeMacas);
17 }
18
19 // Mensagem final
20 printf("Contagem concluída! Total de maçãs: %d\n", numeroDeMacas);
21
22 return 0;
23 }
24 }
```

Figura 1 | Exemplo – algoritmo de contagem.

Neste exemplo, o algoritmo utiliza um *loop while* para simular a ação de contar maçãs. O programa inicia com zero maçãs, e o loop continua enquanto o número de maçãs é menor que 10. A cada iteração do *loop*, o número de maçãs é incrementado, e uma mensagem é exibida indicando quantas maçãs foram contadas até o momento. O processo continua até que 10 maçãs sejam contadas, e uma mensagem final informa o total de maçãs contadas.

Esse exemplo simples ilustra como estruturas de repetição determinísticas, como o *while*, podem ser aplicadas em atividades cotidianas, como contar objetos.

Resumidamente, o histórico das estruturas de repetição determinísticas é um reflexo da constante busca por aprimoramento e eficiência na programação. Desde suas origens humildes até sua integração nas linguagens de programação modernas, essas estruturas têm desempenhado um papel de grande importância na resolução de problemas computacionais complexos.

# ALGORITMOS E TÉCNICAS DE PROGRAMAÇÃO

Sua relevância no mundo da programação reside não apenas em sua capacidade de automatizar tarefas repetitivas, mas também em sua contribuição para a construção de algoritmos eficientes.

## Comparativo com estruturas condicionais

As estruturas determinísticas, também conhecidas como estruturas de repetição definidas, são essenciais na automação de tarefas repetitivas em programas. Por meio dessas estruturas, como *loops "for"* e *"while"*, os programadores podem executar um bloco de código várias vezes com base em uma condição predefinida, proporcionando eficiência e precisão na implementação de algoritmos.

Compreender as estruturas determinísticas é essencial para dominar o controle de fluxo em programas e desenvolver soluções computacionais eficazes.

As estruturas de repetição determinísticas, como o comando *"for"* em linguagens como C e Python, são usadas quando sabemos exatamente quantas vezes um bloco de código precisa ser repetido. Por exemplo, se quisermos imprimir os números de 1 a 5, podemos usar um *loop "for"* que itera cinco vezes, observe na Figura 2:



```
4
5 √ for (int i = 1; i <= 5; i++) {
6 printf("%d\n", i);
7 }
8
```

Figura 2 | Exemplo de estrutura determinística em C.

De acordo com a Figura 2 podemos perceber que as estruturas de repetição determinísticas permitem a execução repetida de um conjunto de instruções com base em uma condição predefinida.

Por outro lado, as estruturas condicionais desempenham um papel importante na tomada de decisões em programas, permitindo que diferentes blocos de código sejam executados com base em condições específicas.

Utilizando comandos como *"if"*, *"else if"* e *"else"*, os programadores podem direcionar o fluxo de execução do programa de acordo com as necessidades do problema em questão. Por exemplo, se quisermos verificar se um número é par ou ímpar, podemos usar uma estrutura condicional como a demonstrada na Figura 3:

# ALGORITMOS E TÉCNICAS DE PROGRAMAÇÃO

```
5 int numero = 10;
6
7 if (numero % 2 == 0) {
8 printf("O número é par.\n");
9 } else {
10 printf("O número é ímpar.\n");
11 }
12
```

Figura 3 | Exemplo de estrutura condicional em C.

A estrutura condicional foi utilizada neste código para determinar se o número armazenado na variável "número" é par ou ímpar. A condição `número % 2 == 0` verifica se o resto da divisão de "número" por 2 é igual a zero.

Se isso for verdadeiro, significa que o número é par e a mensagem "O número é par." é impressa na tela. Caso contrário, se o resto da divisão não for zero, isso indica que o número é ímpar, e a mensagem "O número é ímpar" é impressa na tela. Portanto, a estrutura condicional permite que o programa tome decisões e exiba mensagens diferentes com base no valor da variável "número".

Para que a compreensão acerca desses dois tipos de repetições fique mais clara, abordaremos a seguir três exemplos em que podem ser aplicadas em diversas situações do nosso cotidiano.

## 1. Controle de estoque em um supermercado:

- **Estrutura determinística:** para realizar a contagem do estoque de produtos em um supermercado, pode-se utilizar um *loop "for"* para percorrer todas as prateleiras e contar a quantidade de cada item.
- **Estrutura condicional:** se a quantidade de um determinado produto estiver abaixo de um limite mínimo estabelecido, um alerta pode ser acionado para informar a necessidade de reabastecimento desse item.

## 2. Sistema de alarme residencial:

- **Estrutura determinística:** um *loop "while"* pode ser usado para monitorar continuamente os sensores de movimento e portas em uma casa, verificando se há alguma atividade suspeita.

# ALGORITMOS E TÉCNICAS DE PROGRAMAÇÃO

- **Estrutura condicional:** se um sensor de movimento detectar movimento dentro da casa enquanto o sistema de alarme estiver armado, um sinal de alerta pode ser emitido e as autoridades podem ser contatadas automaticamente.

### 3. Aplicativo de calendário:

- **Estrutura determinística:** para exibir os eventos de um calendário em uma interface de usuário, pode-se usar um *loop "for"* para percorrer todos os eventos e exibi-los na tela.
- **Estrutura condicional:** se um evento estiver agendado para o dia atual, ele pode ser destacado ou exibido de forma diferente para chamar a atenção do usuário. Além disso, se um evento for uma reunião importante, o aplicativo pode enviar uma notificação para lembrar o usuário com antecedência.

Esses exemplos ilustram como as estruturas determinísticas e condicionais são aplicadas em situações do cotidiano para automatizar tarefas, tomar decisões e melhorar a eficiência dos sistemas.

Concluindo, as estruturas de repetição determinísticas são usadas quando precisamos repetir um bloco de código um número específico de vezes, enquanto as estruturas condicionais são usadas para controlar o fluxo de execução com base em condições específicas.

Ambos os tipos de estruturas são essenciais na construção de algoritmos e na resolução de problemas computacionais, e entender suas diferenças e aplicações é fundamental para se tornar um programador que domine esses dois tipos de habilidade.

**Siga em Frente...**

## Aplicações das estruturas de repetição determinísticas

A compreensão das aplicações das estruturas de repetição determinísticas é essencial para qualquer programador, independentemente do nível de experiência. Essas estruturas oferecem uma maneira poderosa de automatizar tarefas repetitivas em programas, o que pode resultar em uma maior produtividade no desenvolvimento de software.

No nosso dia a dia usamos esse tipo de estrutura muitas vezes até sem perceber, para ficar claro e para que você possa visualizar na prática qual é essência desse tipo de estrutura observe como as aplicações das estruturas de repetição determinísticas podem ser úteis em situações do dia a dia por meio dos seguintes exemplos:

- **Preparação de refeições:** ao seguir uma receita para preparar uma refeição, podemos encontrar várias etapas que são repetidas várias vezes. Em uma receita de bolo, por

# ALGORITMOS E TÉCNICAS DE PROGRAMAÇÃO

exemplo, podemos ter que bater a massa, adicionar ingredientes, misturar e assar várias vezes.

Nesse caso, podemos usar uma estrutura de repetição determinística para iterar sobre cada etapa da receita, garantindo que cada passo seja seguido corretamente e resulte em uma refeição deliciosa.

- **Limpeza da casa:** ao realizar a limpeza da casa, muitas vezes nos deparamos com tarefas que precisam ser repetidas em diferentes áreas. Por exemplo, varrer o chão, limpar os móveis, aspirar os tapetes e lavar a louça são atividades que podem ser repetidas em cada cômodo da casa. Usando uma estrutura de repetição determinística, podemos criar um plano de limpeza que nos guie em cada etapa do processo, garantindo uma limpeza completa e eficiente.
- **Exercícios físicos:** durante uma sessão de exercícios físicos, é comum realizar uma série de movimentos repetitivos para fortalecer os músculos e melhorar a resistência. Como em uma rotina de musculação, em que podemos executar várias repetições de levantamento de pesos ou flexões. Utilizando uma estrutura de repetição determinística, podemos programar o número de repetições e conjuntos que desejamos realizar, permitindo-nos acompanhar nosso progresso e alcançar nossos objetivos de condicionamento físico.
- **Estudos e revisões:** ao estudar para um exame ou revisar um conteúdo importante, é comum revisar o material várias vezes para reforçar o aprendizado. Por exemplo, podemos criar um plano de estudos que inclua revisões diárias de conceitos-chave e resolução de exercícios práticos. Utilizando uma estrutura de repetição determinística, podemos agendar essas revisões de forma sistemática, garantindo uma preparação eficaz e um melhor desempenho acadêmico.

Uma das principais áreas em que as estruturas de repetição determinísticas são amplamente aplicadas é no processamento de dados. Por exemplo, ao trabalhar com grandes conjuntos de dados em ciência de dados ou análise estatística, essas estruturas podem ser utilizadas para iterar sobre os dados, aplicar transformações ou realizar cálculos complexos de forma eficiente.

Isso permite que os programadores automatizem processos maçantes e concentrem seu tempo e energia em análises mais avançadas e na extração de informações úteis dos dados. Observe alguns exemplos em que esse tipo de estrutura pode ser aplicado na prática:

- **Processamento de folhas de pagamento:** em um departamento de recursos humanos, as estruturas de repetição determinísticas são utilizadas para processar folhas de pagamento. Por exemplo, ao calcular os salários dos funcionários, um loop pode ser usado para percorrer a lista de funcionários e aplicar os cálculos de salário a cada um deles. Isso permite automatizar o processo e garantir que todos os funcionários sejam pagos corretamente a cada período de pagamento.
- **Atualização de estoque em um supermercado:** em um supermercado, as estruturas de repetição determinísticas são empregadas no processo de atualização de estoque. Por exemplo, ao receber um novo lote de produtos, um loop pode ser usado para percorrer a

# ALGORITMOS E TÉCNICAS DE PROGRAMAÇÃO

lista de itens recebidos e atualizar as quantidades em estoque para cada produto. Isso garante que o estoque seja mantido atualizado e preciso, facilitando o gerenciamento de inventário.

- **Processamento de transações financeiras:** em um banco ou instituição financeira, as estruturas de repetição determinísticas são utilizadas no processamento de transações financeiras. Por exemplo, ao reconciliar transações bancárias, um loop pode ser usado para percorrer a lista de transações e realizar operações como verificação de saldo, débito ou crédito em cada conta. Isso permite automatizar o processo de reconciliação e garantir a precisão das informações financeiras.
- **Análise de dados de vendas:** em uma empresa de varejo, as estruturas de repetição determinísticas são aplicadas na análise de dados de vendas. Ao analisar os dados de vendas de um determinado período, um loop pode ser usado para percorrer a lista de transações e calcular métricas como vendas totais, média de vendas por dia ou produtos mais vendidos. Isso facilita a identificação de padrões de vendas e a tomada de decisões estratégicas para impulsionar os negócios.

Esses exemplos ilustram como as estruturas de repetição determinísticas são aplicadas no processamento de dados em situações do cotidiano, automatizando tarefas repetitivas e facilitando a análise e gerenciamento de informações.

Além disso, as estruturas de repetição determinísticas são fundamentais em muitas áreas da engenharia de software, como no desenvolvimento de algoritmos de busca e ordenação, na manipulação de imagens ou na simulação de sistemas complexos. Ao entender as aplicações dessas estruturas, você será capaz de projetar e implementar soluções mais eficientes para uma série de problemas computacionais.

Portanto, dedicar-se ao estudo das estruturas de repetição determinísticas é fundamental para qualquer estudante ou profissional da área de programação, pois oferecem maneiras eficientes de lidar com tarefas repetitivas no processamento de dados, o que é essencial em um conjunto de aplicações, desde o desenvolvimento de software até análise de dados e automação de processos.

Ao dominá-las, você estará equipado com uma habilidade fundamental que o capacitará a criar programas mais completos e de qualidade, além de prepará-lo para enfrentar desafios cada vez mais complexos no mundo da computação. Portanto, dedicar tempo e esforço ao estudo dessas estruturas é um investimento valioso que certamente renderá frutos ao longo de toda a carreira profissional.

## Vamos Exercitar?

Descrição da situação-problema:

# ALGORITMOS E TÉCNICAS DE PROGRAMAÇÃO

Um *e-commerce* deseja disponibilizar os produtos de sua loja na página de busca de produtos. Para isso, é necessário criar um algoritmo utilizando o comando *while* na linguagem de programação C.

## Resolução da situação-problema:

### Identificação dos Dados:

1. **Inicialização das variáveis:** Defina as variáveis necessárias para o algoritmo, como contador, limite de produtos por página e a lista de produtos disponíveis no *e-commerce*.
2. **Verificação da condição:** utilize o comando *while* para verificar se ainda existem produtos para serem disponibilizados na página de busca.
3. **Exibição dos produtos:** dentro do laço *while*, exiba os produtos na página de busca, respeitando o limite de produtos por página.
4. **Atualização do contador:** após exibir os produtos, atualize o contador para indicar que esses produtos já foram disponibilizados na página de busca.
5. **Condição de parada:** Verifique se todos os produtos já foram exibidos, caso contrário, retorne ao passo 3.
6. **Fim do algoritmo:** ao final do processo, encerre o algoritmo.

Com esse algoritmo, o *e-commerce* poderá disponibilizar os produtos de forma eficiente e organizada na página de busca, garantindo uma melhor experiência para os usuários que estão em busca de produtos específicos.

## Saiba mais

Para saber mais sobre as estruturas de repetição determinísticas, seus principais fundamentos, suas características e, ainda, continuar se aprofundando no assunto, acesse em sua Biblioteca Virtual o livro [Elementos de Programação em C](#), Capítulo 8 - páginas 171 a 190.

## Referências

CORMEN, T. **Algoritmos – Teoria e Prática**. 3.ed. Rio de Janeiro: LTC, 2022.

CORMEN, T. **Desmistificando Algoritmos**. Grupo GEN, 2013. E-book.

FERREIRA, Aurélio Buarque de Holanda. **Novo Aurélio Século XXI**: o dicionário da língua portuguesa. 3 ed. totalmente rev. e ampl. Rio de Janeiro: Nova Fronteira, 1999.

MANZANO, J.A. N G.; OLIVEIRA, J. F. de. **Estudo Dirigido de Algoritmos**. 15. ed. Editora Saraiva, 2012. E-book.

# ALGORITMOS E TÉCNICAS DE PROGRAMAÇÃO

MENÉNDEZ, A. **Simplificando Algoritmos**. Grupo GEN, 2023. E-book.

PINHEIRO, F. de A. C. **Elementos de Programação em C**. Porto Alegre: Bookman, 2012.

SERPA, M. S.; RODRIGUES, T. N.; ALVES, I. C.; *et al.* **Análise de Algoritmos**. Grupo A, 2021. E-book.

WAZLAWICK, R. S. **Introdução a Algoritmos e Programação com Python: Uma Abordagem Dirigida por Testes**. Rio de Janeiro: LTC, 2018.

## Aula 5

Encerramento da Unidade

### Videoaula de Encerramento

#### Este conteúdo é um vídeo!



Para assistir este conteúdo é necessário que você acesse o AVA pelo computador ou pelo aplicativo. Você pode baixar os vídeos direto no aplicativo para assistir mesmo sem conexão à internet.

#### Dica para você

Aproveite o acesso para baixar os slides do vídeo, isso pode deixar sua aprendizagem ainda mais completa.

Olá, estudante! Nessa videoaula, vamos mergulhar fundo nas "condicionais" ou condições e os algoritmos dinâmicos que podemos construir por meio delas. A compreensão dos conteúdos presentes na unidade de ensino é indispensável para qualquer profissional no âmbito de se especializar na área de desenvolvimento de sistemas.

Então, embarque em mais este aprendizado, consistente no cenário de algoritmos. Assista agora, confira a aula que foi pensada e preparada para você, estudante, e dê mais um passo em busca do conhecimento.

### Ponto de Chegada

Olá, estudante! Para desenvolver a competência desta Unidade, que é "Identificar e compreender a estrutura de decisão e a estrutura de repetição", devemos, primeiramente, conhecer os tipos de

# ALGORITMOS E TÉCNICAS DE PROGRAMAÇÃO

dados primitivos, principais comandos de entrada e saída de dados e a estrutura de um programa em uma linguagem de programação.

Entender a lógica por trás das estruturas condicionais e de repetição é necessário para a construção de programas que atendam às necessidades específicas de diferentes contextos.

No âmbito das estruturas de decisão, o estudante aprenderá a utilizar comandos como "if", "else" e "switch" para realizar escolhas lógicas em seus algoritmos. Você também compreenderá como implementar condições que orientam o fluxo do programa, tomando decisões fundamentais para o processamento de dados.

No que se refere às estruturas de repetição, o estudante será capacitado a empregar comandos como "while", "do-while" e "for" para criar iterações controladas, permitindo a automação de tarefas repetitivas. Entenderá como otimizar o uso dessas estruturas, contribuindo para a eficiência e simplicidade de seus algoritmos.

A competência de identificar e compreender essas estruturas proporcionará ao estudante uma base sólida para a resolução de problemas complexos, uma vez que muitas situações do mundo real demandam algoritmos que saibam tomar decisões e realizar repetições de maneira inteligente. Essa habilidade é essencial para quem busca se destacar na programação e na solução de desafios computacionais.

Assim, a identificação e compreensão dessas estruturas são alicerce para a construção de algoritmos sofisticados e eficazes. Essas habilidades não capacitam o estudante a enfrentar desafios práticos na programação, preparando-o para abordar questões complexas em diversas áreas da computação. O desenvolvimento dessa competência é, portanto, um grande passo na formação de um programador habilidoso e criativo.

## É Hora de Praticar!

Um banco deseja adicionar a funcionalidade de investimentos simples em seu aplicativo, permitindo que os usuários possam investir um valor através de sua conta. O objetivo é proporcionar uma opção de investimento fácil e acessível para os clientes do banco.

O "desafio" é desenvolver a funcionalidade de investimento utilizando técnicas algorítmicas avançadas de acordo com os requisitos exigidos.

A sugestão de passos para a criação de uma solução dessa funcionalidade está descrita a seguir..

- Identificação do Requisito.
- Análise de Viabilidade.
- Design da Funcionalidade.
- Desenvolvimento e Testes.
- Integração com Sistemas Existentes.

# ALGORITMOS E TÉCNICAS DE PROGRAMAÇÃO

- Lançamento e Comunicação.
- Monitoramento e Melhoria Contínua.
- Considere a importância das estruturas de seleção na programação. Reflita sobre como essas estruturas permitem que um programa faça escolhas dinâmicas com base em condições específicas.
- Pense em situações do cotidiano em que a capacidade de tomar decisões é essencial. Como essa capacidade se traduziria em termos de programação? Como as estruturas de seleção podem ser aplicadas para resolver problemas do mundo real?
- Agora, direcione sua reflexão para os laços de repetição. Explore como essas estruturas possibilitam a execução repetida de um conjunto de instruções. Considere situações em que a repetição é uma parte fundamental do processo. Pense em tarefas que envolvem a execução de uma ação várias vezes e como os laços de repetição facilitam essa repetição de forma eficiente. Como a capacidade de criar loops pode simplificar a resolução de problemas na programação?
- Explore as nuances desses conceitos e descubra o poder que eles oferecem para transformar sua habilidade na programação. O conhecimento dessas estruturas é a chave para desbloquear novas possibilidades em sua jornada como programador.

## Sugestão de solução

### Passo 1: Identificação do requisito:

- Realizar uma reunião com a equipe de desenvolvimento de software e com os *stakeholders* para entender completamente os requisitos e expectativas em relação à funcionalidade de investimentos simples.

### Passo 2: Análise de viabilidade:

- Avaliar a viabilidade técnica e financeira da implementação da funcionalidade de investimentos simples no aplicativo do banco.
- Considerar os requisitos de segurança e conformidade regulatória para garantir a proteção dos dados e transações dos usuários.

### Passo 3: Design da funcionalidade:

- Criar um design detalhado da interface do usuário para a funcionalidade de investimentos simples, garantindo que seja intuitiva e fácil de usar.
- Definir os fluxos de navegação e interação do usuário durante o processo de investimento.

### Passo 4: Desenvolvimento e testes:

- Implementar a funcionalidade de investimentos simples no aplicativo, seguindo as melhores práticas de desenvolvimento de software.
- Realizar testes rigorosos para garantir que a funcionalidade esteja livre de erros e atenda às expectativas dos usuários.

# ALGORITMOS E TÉCNICAS DE PROGRAMAÇÃO

## Passo 5: Integração com sistemas existentes:

- Integrar a funcionalidade de investimentos simples com os sistemas existentes do banco, como o sistema de contas dos usuários e o sistema de transações financeiras.

## Passo 6: Lançamento e comunicação:

- Lançar a funcionalidade de investimentos simples no aplicativo, comunicando claramente aos usuários sobre essa nova opção de investimento.
- Fornece o suporte e orientação aos usuários que desejam utilizar a funcionalidade de investimentos simples.

## Passo 7: Monitoramento e melhoria contínua:

- Estabelecer um sistema de monitoramento para acompanhar o desempenho da funcionalidade de investimentos simples e coletar feedback dos usuários.
- Utilizar as informações coletadas para realizar melhorias contínuas na funcionalidade, visando sempre proporcionar a melhor experiência para os usuários.

## Conclusão

Com a implementação bem-sucedida da funcionalidade de investimentos simples, o banco poderá oferecer aos seus clientes uma forma conveniente e segura de investir, fortalecendo assim o relacionamento com sua base de clientes.

Nesta linha do tempo, apresentamos resumidamente os principais conceitos relacionados a: estruturas de decisão e laços de repetições e como eles são utilizados na programação. Todos esses, são tópicos fundamentais para a disciplina de Algoritmos e Técnicas de Programação. Convidamos você estudante, a refletir sobre os conceitos apresentados.

# ALGORITMOS E TÉCNICAS DE PROGRAMAÇÃO

## ESTRUTURAS DE DECISÃO E REPETIÇÃO

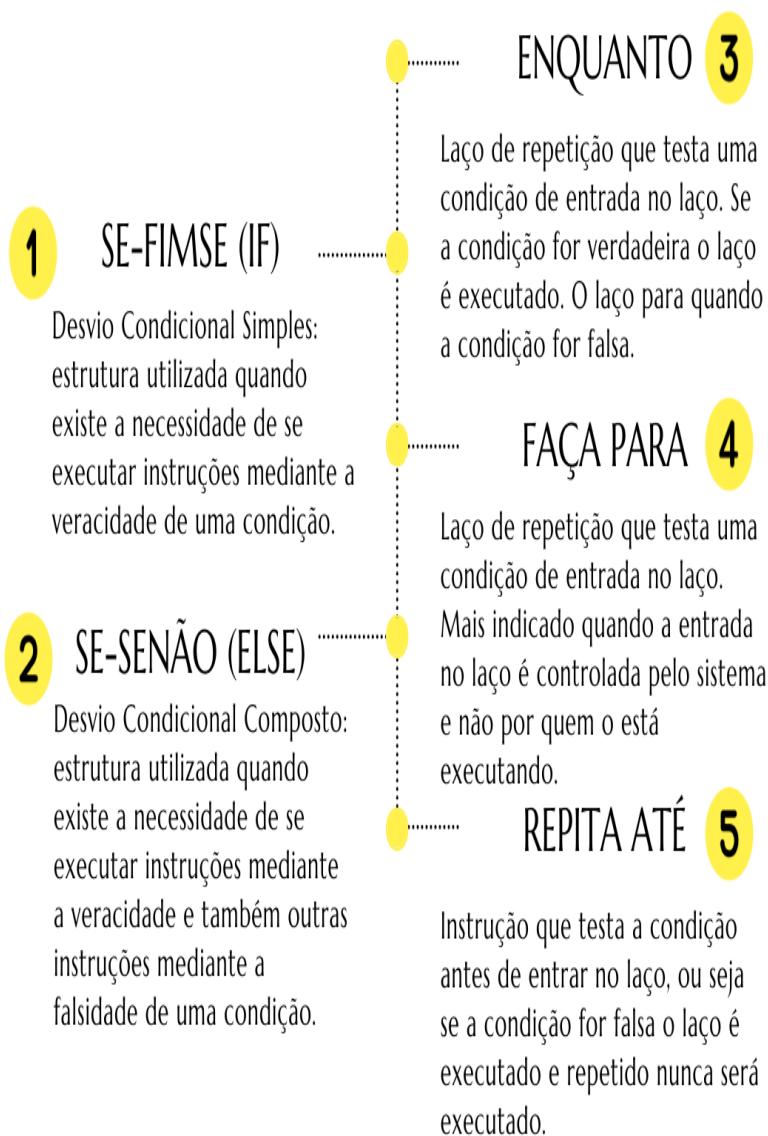


Figura | Estruturas de Decisão e Repetição

CORMEN, T. **Algoritmos** – Teoria e Prática. 3.ed. Rio de Janeiro: LTC, 2022.

# ALGORITMOS E TÉCNICAS DE PROGRAMAÇÃO

CORMEN, T. **Desmistificando Algoritmos.** [S. I.]: Grupo GEN, 2013. E-book.

FERREIRA, A. B. H. **Novo Aurélio Século XXI:** o dicionário da língua portuguesa. 3 ed. Rio de Janeiro: Nova Fronteira, 1999.

MANZANO, J. A. N. G.; OLIVEIRA, J. F. **Estudo Dirigido de Algoritmos.** 15. ed. Editora Saraiva, 2012. E-book.

MENÉNDEZ, A. **Simplificando Algoritmos.** [S. I.]: Grupo GEN, 2023.

PINHEIRO, F. A. C. **Elementos de Programação em C.** Porto Alegre: Bookman, 2012.

SERPA, M. S.; et al. **Análise de Algoritmos.** [S. I.]: Grupo A, 2021.

WAZLAWICK, R. S. **Introdução a Algoritmos e Programação com Python: Uma Abordagem Dirigida por Testes.** Rio de Janeiro: LTC, 2018.

## Unidade 4

### Funções e Recursividade

#### Aula 1

##### Procedimentos e Funções

#### Procedimentos e funções

##### Este conteúdo é um vídeo!



Para assistir este conteúdo é necessário que você acesse o AVA pelo computador ou pelo aplicativo. Você pode baixar os vídeos direto no aplicativo para assistir mesmo sem conexão à internet.

Dica para você

Aproveite o acesso para baixar os slides do vídeo, isso pode deixar sua aprendizagem ainda mais completa.

Olá, estudante! É com grande entusiasmo que convido você a participar da videoaula de abertura desta unidade de ensino, que se dedicará ao aprofundamento nos temas de procedimentos e funções em programação. Estamos empenhados em oferecer um conteúdo de qualidade, alinhado às competências e habilidades que são essenciais para o seu desenvolvimento na área. Nesta videoaula, exploraremos conceitos fundamentais relacionados à definição de procedimentos e funções, proporcionando uma compreensão sólida sobre como estruturar e

# ALGORITMOS E TÉCNICAS DE PROGRAMAÇÃO

organizar o código de maneira eficiente. Abordaremos o uso de funções com ponteiros, destacando a importância dessa técnica na manipulação eficiente de dados na memória.

Além disso, dedicaremos tempo para discutir as funções com e sem retorno, elucidando suas características e aplicações práticas. Para tornar o aprendizado mais envolvente, apresentaremos exemplos concretos que demonstrarão a relevância e a aplicabilidade desses conceitos no desenvolvimento de programas.

Acreditamos que os conhecimentos adquiridos nesta aula não apenas ampliarão sua compreensão sobre procedimentos e funções, mas também contribuirão para fortalecer sua capacidade de pensamento crítico e resolução de problemas na programação.

Não perca a oportunidade de participar ativamente desta jornada de aprendizado. Explore os conceitos apresentados, envolva-se nas discussões e desafios propostos. Sua participação é fundamental na construção do seu conhecimento e no desenvolvimento das habilidades necessárias para se destacar na área de programação.

Vamos lá?

## Ponto de Partida

Olá, estudante! Nesta aula, exploraremos procedimentos e funções, elementos essenciais para qualquer programador que almeje agregar seu conhecimento. Vamos direcionar nosso foco para a definição de procedimentos e funções, abordando a importância de estruturar o código de forma eficaz.

A compreensão do uso de funções com ponteiros será um ponto central em nossa discussão, oferecendo insights valiosos sobre como manipular dados na memória de maneira mais eficiente. Este é um conceito fundamental para programadores que buscam otimizar seus códigos e melhorar a performance de suas aplicações.

Além do mais, exploraremos as funções com e sem retorno, destacando suas características e aplicações práticas. Com exemplos concretos, pretende-se ilustrar como esses elementos podem ser integrados de maneira eficiente em programas, proporcionando resultados desejados.

Apresentaremos, ao final, uma situação-problema com um algoritmo que por meio de uma função valide se um número é positivo ou negativo.

Convidamos você a mergulhar nesse conteúdo, dedicando-se à construção de uma base sólida para sua jornada na programação.

Esteja preparado para uma imersão profunda nesses conceitos fundamentais. Sua participação ativa e envolvimento serão peças-chave na construção do seu conhecimento e no desenvolvimento das habilidades necessárias para se destacar na programação.

Vamos juntos nessa jornada de aprendizado!

# ALGORITMOS E TÉCNICAS DE PROGRAMAÇÃO

## Vamos Começar!

A programação em linguagens como C é fundamentada em conceitos essenciais, sendo a definição de procedimentos e funções um dos pilares centrais. Procedimentos e funções são blocos de código reutilizáveis que desempenham papéis específicos dentro de um programa, proporcionando uma estrutura modular que facilita o desenvolvimento e a manutenção do código.

A definição de procedimentos e funções permite a organização do código em unidades lógicas e independentes, facilitando a compreensão e a manutenção do software. Ao dividir tarefas em pequenas partes, torna-se mais simples isolar e resolver problemas, promovendo a legibilidade e modularidade do código. Essa abordagem é crucial em projetos de grande escala, onde a complexidade pode ser gerenciada de maneira mais eficiente.

Ao adicionar ponteiros à equação, a capacidade de manipular a memória de forma direta torna-se uma ferramenta poderosa. O uso de funções com ponteiros permite a passagem eficiente de dados entre diferentes partes do programa, possibilitando operações mais avançadas, como a manipulação de estruturas de dados complexas e o acesso direto à memória, proporcionando eficiência e flexibilidade na implementação de algoritmos.

A distinção entre funções com e sem retorno é crucial na programação em C. Funções com retorno fornecem um valor específico ao ponto de chamada, permitindo a comunicação eficaz de resultados.

Por outro lado, funções sem retorno executam ações específicas, sendo essenciais para realizar tarefas sem a necessidade de retornar um valor explícito. Compreender quando e como utilizar cada tipo de função é fundamental para desenvolver código coeso e eficiente.

A definição de procedimentos e funções, o uso de funções com ponteiros e a compreensão das funções com e sem retorno são elementos cruciais para qualquer programador em C.

Estes conceitos não apenas fornecem uma estrutura organizada para o código, mas também possibilitam a implementação de soluções eficientes e flexíveis, contribuindo para o desenvolvimento de software robusto e de alta qualidade.

## Definição de procedimentos e funções

A linguagem de programação C é conhecida por sua eficiência e flexibilidade, sendo amplamente utilizada no desenvolvimento de sistemas e aplicações. Para criar programas mais estruturados e modulares, são essenciais dois conceitos fundamentais: procedimentos e funções.

## Procedimentos em C

# ALGORITMOS E TÉCNICAS DE PROGRAMAÇÃO

Procedimentos são blocos de código que realizam uma tarefa específica e não retornam um valor. Eles são definidos pela palavra-chave *void* quando não têm um valor de retorno. Um procedimento pode aceitar parâmetros, que são valores fornecidos durante a chamada do procedimento para personalizar sua execução. Vejamos um exemplo de procedimento em C:

```
1 #include <stdio.h>
2
3
4 √ void saudacao() {
5 printf("Olá, mundo!\n");
6 }
7
8 √ int main() {
9 saudacao(); // Chamando o procedimento
10 return 0;
11 }
12
13
14
```

Figura 1 | Procedimento básico.

## Funções em C

Funções são similares a procedimentos, mas podem retornar um valor. Elas são definidas com um tipo de retorno específico (além de *void*) e podem ter parâmetros para receber dados de entrada. A palavra-chave *return* é usada para enviar um valor de volta ao ponto de chamada. Exemplo de função em C:

# ALGORITMOS E TÉCNICAS DE PROGRAMAÇÃO

```

1 #include <stdio.h>
2
3 int soma(int a, int b) {
4 return a + b;
5 }
6
7 int main() {
8 int resultado = soma(3, 5); // Chamando a função e armazenando o resultado
9 printf("A soma é: %d\n", resultado);
10 return 0;
11 }
12

```

Figura 2 | Função Soma.

## Parâmetros e Retorno em detalhes

Parâmetros permitem que procedimentos e funções recebam valores externos, tornando o código mais flexível. O retorno é útil quando desejamos obter resultados específicos do procedimento ou da função.

## Escopo de variáveis

Variáveis declaradas dentro de um procedimento ou função têm um escopo local, o que significa que são acessíveis apenas dentro desse bloco de código. Isso ajuda a evitar conflitos de nomes e melhora a organização do código.

Procedimentos e funções são componentes cruciais em programação C, facilitando a modularidade e reutilização de código. Ao entender e aplicar esses conceitos, os programadores conseguem criar sistemas mais eficientes, legíveis e fáceis de manter.

## Esqueleto de uma função em C

O esqueleto básico de uma função em C segue o seguinte formato:

```

tipo_retorno nome_da_funcao(tipo_parametro1 parametro1, ...) {
 // Corpo da função
 // Código que realiza uma tarefa específica
 // Retorno, se a função tiver um tipo de retorno
 return valor_de_retorno;
}

```

- **tipo\_retorno**: o tipo de dado que a função retorna, como *int*, *float*, *void* etc.

# ALGORITMOS E TÉCNICAS DE PROGRAMAÇÃO

- **nome\_da\_funcao:** o nome dado à função, seguindo as regras de nomenclatura.
- **(tipo\_parametro1 parametro1, tipo\_parametro2 parametro2, ...):** Parâmetros que a função pode receber. cada parâmetro é composto pelo tipo e pelo nome.
- **// Corpo da função:** o bloco de código que realiza a tarefa da função.
- **return valor\_de\_retorno;**: Se a função tem um tipo de retorno diferente de *void*, a instrução *return* é usada para enviar o valor de volta ao ponto de chamada.

## Esqueleto de um procedimento em C

O esqueleto de um procedimento é semelhante ao de uma função, mas com a diferença de que um procedimento não retorna um valor. O tipo de retorno é substituído por *void* para indicar que a função não retorna nada.

```
void nome_do_procedimento(tipo_parametro1 parametro1, ...) {
 // Corpo do procedimento
 // Código que realiza uma tarefa específica
}
```

Ambos os esqueletos seguem a estrutura geral de uma declaração de função ou procedimento em C, oferecendo flexibilidade para acomodar diferentes necessidades de programação.

## Uso de funções com ponteiros

O uso de funções com ponteiros é uma prática comum em linguagem C, proporcionando maior flexibilidade e eficiência na manipulação de dados. Ponteiros são variáveis que armazenam endereços de memória, permitindo o acesso direto e a modificação dos dados naquela posição.

O esqueleto básico de um ponteiro em C é composto pelo tipo de dado ao qual o ponteiro aponta, seguido pelo operador de declaração de ponteiro (\*), e, finalmente, o nome do ponteiro. A sintaxe geral é a seguinte:

```
tipo_dado *nome_do_ponteiro;
```

- **tipo\_dado:** o tipo de dado ao qual o ponteiro aponta. Pode ser qualquer tipo de dado, como *int*, *float*, *char*, ou até mesmo outro tipo de ponteiro.
- **\***: O operador de declaração de ponteiro, indicando que a variável é um ponteiro.
- **nome\_do\_ponteiro:** o nome atribuído ao ponteiro, seguindo as regras de nomenclatura.

```
int *ptr_numero; // Ponteiro para um inteiro
```

# ALGORITMOS E TÉCNICAS DE PROGRAMAÇÃO

Este esqueleto apenas declara um ponteiro, mas não o inicializa com um endereço de memória específico. A inicialização pode ocorrer posteriormente usando o operador de endereço (&) para obter o endereço de uma variável existente:

```
int numero = 10;
int *ptr_numero = № // Inicializando o ponteiro com o endereço de 'numero'
```

Assim, *ptr\_numero* agora aponta para a localização de memória onde o valor de número está armazenado. O uso adequado de ponteiros é crucial para manipulações de dados eficientes em C, especialmente em situações que envolvem alocação dinâmica de memória e funções que manipulam valores fora de seu escopo original.

## Passagem de ponteiros como parâmetros

Ao passar um ponteiro como parâmetro para uma função, possibilita-se a manipulação direta do conteúdo referenciado por esse ponteiro dentro da função. Isso é especialmente útil quando se deseja modificar o valor de uma variável fora do escopo da função. Vejamos um exemplo:

```
1 #include <stdio.h>
2
3 int* aloca_memoria() {
4 int *ponteiro = (int *)malloc(sizeof(int)); // Alocação dinâmica de memória
5 *ponteiro = 10;
6 return ponteiro;
7 }
8
9 int main() {
10 int *ptr = aloca_memoria(); // Recebe o ponteiro retornado pela função
11 printf("Valor alocado dinamicamente: %d\n", *ptr);
12 free(ptr); // Libera a memória alocada dinamicamente
13 return 0;
14 }
```

Figura 3 | Parâmetro com ponteiro.

## Retorno de ponteiros em funções

Funções também podem retornar ponteiros, permitindo a alocação dinâmica de memória ou o retorno de endereços de variáveis locais. A seguir, um exemplo:

# ALGORITMOS E TÉCNICAS DE PROGRAMAÇÃO

```
1 #include <stdio.h>
2
3 int* aloca_memoria() {
4 int *ponteiro = (int *)malloc(sizeof(int)); // Alocação dinâmica de memória
5 *ponteiro = 10;
6 return ponteiro;
7 }
8
9 int main() {
10 int *ptr = aloca_memoria(); // Recebe o ponteiro retornado pela função
11 printf("Valor alocado dinamicamente: %d\n", *ptr);
12 free(ptr); // Libera a memória alocada dinamicamente
13 return 0;
14 }
```

Figura 4 | Retorno com ponteiro.

## Manipulação de arrays com ponteiros

Ao usar ponteiros, é possível manipular *arrays* de forma eficiente, tanto para leitura quanto para escrita, oferecendo uma alternativa mais direta e eficiente do que a indexação tradicional.  
Exemplo de manipulação de *arrays* com ponteiros:

# ALGORITMOS E TÉCNICAS DE PROGRAMAÇÃO

```
1 #include <stdio.h>
2
3 ∵ void dobrar_elementos(int *arr, int tamanho) {
4 ∵ for (int i = 0; i < tamanho; i++) {
5 arr[i] *= 2;
6 }
7 }
8
9 ∵ int main() {
10 int numeros[] = {1, 2, 3, 4, 5};
11 dobrar_elementos(numeros, 5);
12
13 printf("Array dobrado: ");
14 ∵ for (int i = 0; i < 5; i++) {
15 printf("%d ", numeros[i]);
16 }
17
18 return 0;
19 }
20
21 }
```

Figura 5 | Vetor com ponteiro.

O uso de funções com ponteiros em C é uma prática poderosa, permitindo uma manipulação mais eficiente e direta de dados, além de possibilitar a alocação dinâmica de memória. No entanto, é necessário ter cuidado para evitar vazamentos de memória ou acessos indevidos, garantindo a integridade e segurança do programa.

Siga em Frente...

# ALGORITMOS E TÉCNICAS DE PROGRAMAÇÃO

## Funções com e sem retorno

Veremos agora funções com retorno e sem retorno.

### Funções com retorno

Funções com retorno são cruciais quando há a necessidade de produzir e retornar um valor específico após a conclusão da tarefa. Estas funções são declaradas com um tipo de retorno definido (como *int*, *float*, *char* etc.), indicando o tipo de dado que será devolvido ao ponto de chamada. A estrutura geral de uma função com retorno é a seguinte:

```
tipo_retorno nome_da_funcao(tipo_parametro1 parametro1, tipo_parametro2 parametro2, ...) {
 // Corpo da função
 // Código que realiza uma tarefa específica
 // Retorno, se a função tiver um tipo de retorno
 return valor_de_retorno;
}
```

### Funções sem retorno

Em linguagem C, funções sem retorno são definidas pelo uso da palavra-chave “*void*”. Estas funções são empregadas quando a execução de uma tarefa não demanda a produção de um valor específico para ser utilizado posteriormente no programa.

A ausência de um tipo de retorno, indicado por “*void*”, ressalta que a função está focada na execução de ações ou procedimentos, sem a necessidade de gerar um resultado palpável. A estrutura básica de uma função sem retorno é delineada por:

```
void nome_da_funcao(tipo_parametro1 parametro1, tipo_parametro2 parametro2, ...) {
 // Corpo da função
 // Código que realiza uma tarefa específica
}
```

A escolha entre funções com e sem retorno depende da natureza da tarefa a ser realizada. Funções sem retorno são apropriadas para procedimentos que não geram resultados específicos, enquanto funções com retorno são essenciais quando é necessário calcular e comunicar valores específicos para outras partes do programa.

Essa distinção é fundamental para a estruturação eficiente de programas em C. O programa a seguir realiza a soma de dois números e exibe o resultado usando ambas as formas de funções.

# ALGORITMOS E TÉCNICAS DE PROGRAMAÇÃO

```
1 #include <stdio.h>
2
3 // Função sem retorno (procedimento) para exibir uma mensagem
4 void exibir_mensagem() {
5 printf("Esta é uma função sem retorno.\n");
6 }
7
8 // Função com retorno para somar dois números
9 int somar(int a, int b) {
10 return a + b;
11 }
12
13 int main() {
14 // Chamando a função sem retorno (procedimento)
15 exibir_mensagem();
16
17 // Chamando a função com retorno para somar dois números
18 int resultado = somar(3, 5);
19
20 // Exibindo o resultado da soma
21 printf("A soma é: %d\n", resultado);
22
23 return 0;
24 }
```

Figura 6 | Demonstração de retornos.

Neste exemplo, a função *exibir\_mensagem* é uma função sem retorno (ou procedimento), que apenas exibe uma mensagem na tela. Já a função *somar* é uma função com retorno, que recebe dois parâmetros, realiza a soma e retorna o resultado. Ambas são chamadas no *main()* para ilustrar o uso de funções com e sem retorno em um programa C simples.

A linguagem de programação C é conhecida por sua eficiência e flexibilidade, fundamentada em conceitos essenciais como procedimentos e funções. Procedimentos são blocos de código que realizam tarefas específicas sem retornar valores, enquanto funções, além de realizar operações, podem retornar resultados. Essa distinção oferece uma abordagem modular para o desenvolvimento de software, facilitando a organização e manutenção do código.

# ALGORITMOS E TÉCNICAS DE PROGRAMAÇÃO

Ao introduzir ponteiros, a linguagem C amplia suas capacidades, permitindo o acesso direto à memória. Funções com ponteiros tornam-se ferramentas poderosas para manipular dados de forma eficiente, especialmente quando se trata de alocação dinâmica de memória e manipulação de estruturas complexas. Essa característica única da linguagem contribui significativamente para sua versatilidade.

A escolha entre funções com e sem retorno em C depende da necessidade específica de cada tarefa. Funções sem retorno são ideais para procedimentos que executam ações sem gerar resultados específicos, promovendo uma execução mais leve e direta.

Por outro lado, funções com retorno são fundamentais para calcular valores específicos e comunicá-los eficientemente a outras partes do programa.

Esses conceitos trabalham em conjunto para fornecer um ambiente robusto e estruturado na programação em C. A modularidade proveniente de procedimentos e funções permite a reutilização eficaz de código, enquanto o uso judicioso de ponteiros adiciona uma camada de controle sobre a memória, otimizando a eficiência.

A distinção entre funções com e sem retorno oferece a flexibilidade necessária para atender a uma variedade de requisitos de programação.

A síntese harmoniosa de procedimentos, funções, ponteiros e a escolha criteriosa entre funções com e sem retorno são pilares fundamentais na construção de programas robustos e eficientes em C.

Ao compreender e aplicar esses conceitos, os desenvolvedores têm em mãos ferramentas poderosas para enfrentar desafios complexos, garantindo a criação de software confiável e de alta performance.

## Vamos Exercitar?

### Descrição da situação-problema:

Você foi contratado como **desenvolvedor júnior** em uma empresa de software e seu primeiro trabalho é desenvolver uma função do tipo inteiro que por meio de um valor numérico recebido por parâmetro, teste se esse valor é positivo ou negativo.

### Enunciado:

Durante o desenvolvimento da função você deverá:

- a. Solicitar um valor numérico do usuário diferente de 0 (zero).

# ALGORITMOS E TÉCNICAS DE PROGRAMAÇÃO

- b. Criar uma função com retorno numérico inteiro. Caso o valor for positivo a função deve retornar o valor 1 (um), se for negativo, deve retornar o valor 0 (zero).
- c. O programa deve validar se o valor informado é diferente de 0 (zero), caso contrário, ignore o valor e solicite-o novamente ao usuário.

## Resolução da situação-problema:

Segue uma sugestão do algoritmo resolvido na linguagem de programação C.

```

6 int main() {
7 int valor;
8 do {
9 // Solicitar um valor numérico ao usuário
10 printf("Digite um valor numérico diferente de 0: ");
11 scanf("%d", &valor);
12
13 // Validar se o valor é diferente de 0
14 if (valor != 0) {
15 // Chamar a função e exibir o resultado
16 int resultado = verificarPositivoNegativo(valor);
17 printf("O resultado da função é: %d\n", resultado);
18 } else {
19 // Ignorar o valor e solicitar novamente
20 printf("Valor inválido. Tente novamente.\n");
21 }
22
23 } while (valor == 0); // Continuar solicitando até obter um valor diferente de 0
24
25 return 0;
26 }
27
28 // Definição da função
29 int verificarPositivoNegativo(int numero) {
30 // Verificar se o valor é positivo (retornar 1) ou negativo (retornar 0)
31 if (numero > 0) {
32 return 1;
33 } else {
34 return 0;
35 }
36 }
```

Figura 7 | Algoritmo resolvido.

Neste programa, a função **verificarPositivoNegativo()** retorna 1 se o número é positivo e 0 se é negativo. O programa principal solicita um valor ao usuário, válida se é diferente de 0 e chama a função, repetindo o processo até receber um valor válido.

# ALGORITMOS E TÉCNICAS DE PROGRAMAÇÃO

## Saiba mais

Para saber mais sobre funções e procedimentos, acesse em sua Biblioteca Virtual o livro [Elementos de Programação em C](#), Capítulo 9 - Seção 9.2 Declaração, definição e protótipo - páginas 192 e 193.

## Referências

CORMEN, T. **Algoritmos – Teoria e Prática**. 3.ed. Rio de Janeiro: LTC, 2022.

CORMEN, T. **Desmistificando Algoritmos**. Grupo GEN, 2013. E-book.

FERREIRA, Aurélio Buarque de Holanda. **Novo Aurélio Século XXI**: o dicionário da língua portuguesa. 3 ed. totalmente rev. e ampl. Rio de Janeiro: Nova Fronteira, 1999.

MANZANO, J.A. N G.; OLIVEIRA, J. F. de. **Estudo Dirigido de Algoritmos**. 15. ed. Editora Saraiva, 2012. E-book.

MENÉNDEZ, A. **Simplificando Algoritmos**. Grupo GEN, 2023. E-book.

PINHEIRO, F. de A. C. **Elementos de Programação em C**. Porto Alegre: Bookman, 2012.

SERPA, M. S.; RODRIGUES, T. N.; ALVES, I. C.; *et al.* **Análise de Algoritmos**. Grupo A, 2021. E-book.

WAZLAWICK, R. S. **Introdução a Algoritmos e Programação com Python: Uma Abordagem Dirigida por Testes**. Rio de Janeiro: LTC, 2018.

## Aula 2

Escopo e Passagem de Parâmetros

### Escopo e passagem de parâmetros

**Este conteúdo é um vídeo!**

Para assistir este conteúdo é necessário que você acesse o AVA pelo computador ou pelo aplicativo. Você pode baixar os vídeos direto no aplicativo para assistir mesmo sem conexão à internet.

# ALGORITMOS E TÉCNICAS DE PROGRAMAÇÃO



## Dica para você

Aproveite o acesso para baixar os slides do vídeo, isso pode deixar sua aprendizagem ainda mais completa.

Olá, estudante! Nesta videoaula, vamos explorar o escopo de variáveis e constantes, destacando a importância de compreender onde esses elementos são acessíveis e como isso impacta a estrutura do seu código. Além disso, discutiremos a passagem de parâmetros, analisando como informações são transmitidas para funções e métodos.

Aprofundaremos ainda mais nossos estudos ao abordar a passagem por valor e por referência. Entender a diferença entre esses dois métodos é fundamental para evitar erros comuns e otimizar o desempenho do seu código.

Assim como nas aulas anteriores, nossa abordagem será prática, com exemplos aplicados para ilustrar cada conceito. Isso proporcionará uma compreensão mais profunda e prática, permitindo que você aplique esses conhecimentos de forma eficaz em seus projetos futuros.

Lembre-se de que adquirir esse conjunto de habilidades não apenas ampliará sua compreensão técnica, mas também fortalecerá suas habilidades de pensamento crítico e resolução de problemas.

## Ponto de Partida

Olá, estudante! Nesta aula, direcionaremos nossa atenção para temas pontuais: escopo de variáveis e constantes, passagem de parâmetros e passagem por valor e por referência, aprofundando ainda mais nossos estudos em algoritmos e técnicas de programação.

Assim como você explorou outros conteúdos em aulas anteriores, entenderemos agora como o escopo de variáveis e constantes impactam diretamente a execução do código. Teremos como foco a criação de programas mais robustos e eficientes, utilizando adequadamente esses elementos.

A discussão sobre a passagem de parâmetros nos permitirá compreender como as informações são transmitidas entre diferentes partes do programa. Este conhecimento é necessário para desenvolver funções e métodos que interagem de maneira eficaz, contribuindo para a modularidade e organização do código.

No tópico de passagem de parâmetros por valor e por referência, mergulharemos nas nuances desses métodos, destacando suas aplicações e implicações. Essa compreensão profunda evita armadilhas comuns e otimiza o desempenho do seu código.

Tornaremos o aprendizado mais tangível com exemplos práticos que demonstrarão a aplicação direta desses conceitos. Afinal, a melhor maneira de consolidar o conhecimento é aplicá-lo em situações do mundo real.

# ALGORITMOS E TÉCNICAS DE PROGRAMAÇÃO

Para tanto, será apresentada uma situação-problema propondo a criação por parte do estudante, de uma solução algorítmica para o cálculo do reajuste salarial de um colaborador qualquer de uma empresa fictícia.

Lembre-se de que cada tópico abordado ampliará seu entendimento técnico e fortalecerá suas habilidades analíticas e resolutivas.

Então, vamos lá!

## Vamos Começar!

Vamos abordar conceitos essenciais em programação que têm grande impacto no desenvolvimento de habilidades técnicas. Estamos falando sobre o escopo de variáveis e constantes, passagem de parâmetros e as nuances entre passagem por valor e referência.

O escopo de variáveis e constantes define onde esses elementos podem ser acessados no código. Compreender esse conceito é fundamental para organizar e estruturar programas de maneira eficiente.

A passagem de parâmetros, por sua vez, trata de como as informações são transmitidas entre diferentes partes do programa. Esse conhecimento é crucial para criar funções e métodos que interagem de forma eficaz, contribuindo para a modularidade do código.

Quanto à passagem por valor e por referência, exploraremos como os dados são compartilhados entre diferentes partes do código. A diferença crucial é que, na passagem por valor, os dados são copiados, enquanto na passagem por referência, compartilhamos a referência da variável original. Essa distinção é vital para otimizar o desempenho do código.

Para tornar esses conceitos mais tangíveis, utilizaremos exemplos práticos. A teoria é importante, mas a aplicação direta em situações reais fortalece a compreensão.

## Escopo de variáveis e constantes

O escopo em programação refere-se à região do código onde uma variável ou constante é visível e pode ser acessada. Em C, existem dois tipos principais de escopo: global e local. Variáveis globais são declaradas fora de qualquer função e podem ser acessadas em todo o programa, enquanto variáveis locais são definidas dentro de uma função e só podem ser acessadas dentro dessa função.

### Escopo local

# ALGORITMOS E TÉCNICAS DE PROGRAMAÇÃO

Variáveis locais são aquelas declaradas dentro de uma função. Elas só existem enquanto a função está em execução e são destruídas quando a função é concluída. Considere o exemplo a seguir:

```
1 #include <stdio.h>
2
3 void exemploFuncao() {
4 int x = 10; // Variável local
5 printf("%d\n", x);
6 }
7
8 int main() {
9 exemploFuncao();
10 // printf("%d\n", x); // Erro! x não é visível aqui
11 return 0;
12 }
13
```

Figura 1 | Local.

## Escopo global

Variáveis globais, por outro lado, são acessíveis de qualquer lugar no programa. Elas são declaradas fora de funções e mantêm seu valor durante toda a execução. No entanto, o uso excessivo de variáveis globais deve ser evitado para garantir um código mais modular e fácil de entender.

## Constantes

As constantes são valores que não podem ser alterados durante a execução do programa. Em C, podemos usar a palavra-chave *const* para definir constantes. Elas são frequentemente usadas para representar valores fixos que não devem ser modificados. Veja o exemplo:

# ALGORITMOS E TÉCNICAS DE PROGRAMAÇÃO

```
1 #include <stdio.h>
2
3 const int PI = 3.14159; // Constante global
4
5 int main() {
6 printf("O valor de PI é: %f\n", PI);
7 // PI = 3.0; // Erro! Tentativa de modificar uma constante
8 return 0;
9 }
```

Figura 2 | Constantes.

## Escopo de bloco

Em C, o escopo de bloco refere-se à região delimitada por chaves {}. Variáveis locais podem ter escopo de bloco quando declaradas dentro de um bloco específico. Isso limita a visibilidade da variável a esse bloco em particular.

## Ocultação de variáveis

Quando uma variável local tem o mesmo nome que uma variável global, a variável local "oculta" a variável global dentro do escopo local. Isso é conhecido como sombreamento de variáveis. Veja o exemplo:

```
1 #include <stdio.h>
2
3 int x = 5; // Variável global
4
5 void exemploFuncao() {
6 int x = 10; // Variável local que oculta a variável global
7 printf("%d\n", x); // Imprime o valor local (10)
8 }
9
10 int main() {
11 exemploFuncao();
12 printf("%d\n", x); // Imprime o valor global (5)
13 return 0;
14 }
```

# ALGORITMOS E TÉCNICAS DE PROGRAMAÇÃO

Figura 3 | Variável oculta.

## Modificadores de acesso

Em C, é possível usar os modificadores de acesso *extern* e *static* para controlar a visibilidade de variáveis globais. A palavra-chave *extern* permite que uma variável seja usada em vários arquivos, enquanto *static* restringe a variável a um único arquivo.

## Variáveis automáticas

Variáveis locais são, por padrão, automáticas em C, o que significa que são alocadas automaticamente quando uma função é chamada e desalocadas quando a função é concluída. Isso contrasta com variáveis estáticas, que mantêm seu valor entre chamadas de função.

## Variáveis estáticas

Variáveis estáticas mantêm seu valor entre chamadas de função e têm escopo local. Elas são declaradas usando a palavra-chave *static*. O exemplo abaixo ilustra o conceito:

```
1 #include <stdio.h>
2
3 void exemploFuncao() {
4 static int contador = 0; // Variável estática
5 contador++;
6 printf("Contador: %d\n", contador);
7 }
8
9 int main() {
10 exemploFuncao(); // Contador: 1
11 exemploFuncao(); // Contador: 2
12 return 0;
13 }
14
```

Figura 4 | Variável estática.

# ALGORITMOS E TÉCNICAS DE PROGRAMAÇÃO

Ao lidar com escopo de variáveis, é crucial aderir a boas práticas de programação. Evite o uso excessivo de variáveis globais, opte por variáveis locais sempre que possível, e escolha nomes significativos para melhorar a legibilidade do código. O entendimento adequado do escopo é fundamental para criar programas robustos e fáceis de manter.

## Siga em Frente...

### Passagem de parâmetros

A passagem de parâmetros em programação refere-se ao mecanismo pelo qual valores são transmitidos entre partes diferentes de um programa, especialmente entre a chamada de uma função e a própria função. Isso é essencial para permitir a comunicação e manipulação de dados em um programa de maneira eficiente. Existem dois principais métodos de passagem de parâmetros: por valor e por referência.

No método de passagem por valor, uma cópia do valor da variável é transmitida para a função, e qualquer alteração feita dentro da função não afeta a variável original fora dela. Isso é comum para tipos de dados simples, como inteiros e caracteres.

Por outro lado, na passagem por referência, o endereço de memória da variável é transmitido, permitindo que a função acesse e modifique diretamente o conteúdo da variável original. Isso é frequentemente implementado por meio do uso de ponteiros.

A escolha entre passagem por valor e por referência depende das necessidades específicas de um programa. A passagem por valor é mais segura em termos de preservar a integridade dos dados originais, enquanto a passagem por referência é mais eficiente quando se lida com grandes conjuntos de dados e é necessário modificar diretamente as variáveis originais.

Compreender esses métodos é fundamental para o desenvolvimento de código eficiente e modular em linguagens de programação como C.

### Passagem por valor e por referência

A passagem de parâmetros em C é um conceito fundamental que influencia diretamente na forma como as funções interagem com variáveis. No método de passagem por valor, uma cópia do valor da variável é transmitida para a função. Por exemplo, considere a seguinte função em C:

# ALGORITMOS E TÉCNICAS DE PROGRAMAÇÃO

```

1 #include <stdio.h>
2
3 void exemploPassagemPorValor(int x) {
4 x = x * 2; // Altera apenas a cópia local de x
5 printf("Dentro da função: %d\n", x);
6 }
7
8 int main() {
9 int numero = 5;
10 exemploPassagemPorValor(numero);
11 printf("Fora da função: %d\n", numero); // Permanece 5, pois a função atua na cópia
12 return 0;
13 }
```

Figura 5 | Passagem por valor.

Por outro lado, a passagem por referência é implementada utilizando ponteiros em C. Considere o exemplo a seguir:

```

1 #include <stdio.h>
2
3 void exemploPassagemPorReferencia(int *y) {
4 (*y) = (*y) * 2; // Altera diretamente o valor da variável original
5 printf("Dentro da função: %d\n", *y);
6 }
7
8 int main() {
9 int numero = 5;
10 exemploPassagemPorReferencia(&numero);
11 printf("Fora da função: %d\n", numero); // Agora, o valor é modificado para 10
12 return 0;
13 }
```

Figura 6 | Passagem por referência.

Nesse caso, o ponteiro permite que a função modifique diretamente a variável original, proporcionando uma forma eficiente de manipulação de dados. Esses exemplos ilustram as diferenças práticas entre passagem por valor e por referência em C, destacando a importância de escolher o método apropriado com base nos requisitos específicos de cada situação.

Além disso, é importante mencionar que a passagem por valor é mais segura em termos de prevenção contra modificações indesejadas nos dados originais. Isso porque a função recebe uma cópia isolada dos valores, garantindo a integridade das variáveis fora do escopo da função.

No entanto, em casos em que a eficiência e a necessidade de modificar os valores originais são prioridades, a passagem por referência usando ponteiros é uma escolha mais adequada.

# ALGORITMOS E TÉCNICAS DE PROGRAMAÇÃO

É válido destacar que a manipulação de ponteiros requer atenção especial para evitar erros, como diferenciar um ponteiro nulo ou acessar áreas de memória não alocadas. Códigos bem estruturados e documentados são essenciais para garantir a compreensão adequada e a manutenção futura do código, especialmente ao lidar com passagem por referência.

A passagem de parâmetros é um aspecto crucial na construção de programas eficientes e modularizados em C. A compreensão das nuances entre passagem por valor e por referência é vital para tomar decisões informadas ao projetar funções e procedimentos, levando em consideração tanto a segurança quanto o desempenho do código.

## Conclusão

Compreender os conceitos de escopo de variáveis e constantes, passagem de parâmetros e as diferenças entre passagem por valor e por referência é essencial para programadores que buscam escrever código eficiente, modular e de fácil manutenção em linguagens como C.

O entendimento do escopo de variáveis e constantes permite aos desenvolvedores controlar a visibilidade e a acessibilidade de dados em diferentes partes de um programa. A distinção entre variáveis globais e locais, assim como a aplicação adequada de modificadores de acesso como "extern" e "static", contribuem para a estruturação organizada do código, promovendo a modularidade e evitando conflitos indesejados.

Ao explorar a passagem de parâmetros, os programadores ganham a capacidade de transmitir informações entre diferentes partes do código de maneira eficiente. A passagem por valor, onde cópias dos dados são enviadas para funções, e a passagem por referência, usando ponteiros para acessar diretamente a memória, oferecem abordagens distintas para manipulação de dados, cada uma com suas vantagens e considerações específicas.

A passagem por valor é segura e evita modificações indesejadas nas variáveis originais, enquanto a passagem por referência é mais eficiente ao lidar com grandes conjuntos de dados, permitindo modificações diretas nos valores originais.

A escolha entre essas abordagens depende das necessidades específicas do programa, destacando a importância de uma análise cuidadosa ao projetar funções e procedimentos.

No cenário prático, a combinação desses conhecimentos promove o desenvolvimento de código claro, conciso e de fácil manutenção. Boas práticas, como evitar o uso excessivo de variáveis globais, escolher nomes significativos e documentar adequadamente o código, complementam o entendimento dos conceitos fundamentais de programação, consolidando uma base sólida para o desenvolvimento de software robusto e eficiente em C.

## Vamos Exercitar?

# ALGORITMOS E TÉCNICAS DE PROGRAMAÇÃO

## Descrição da situação-problema:

Você, como estudante de programação, foi designado para criar uma solução algorítmica que calcule o reajuste salarial de um colaborador da empresa "ABC001". O objetivo é desenvolver um programa eficiente que leve em consideração o salário atual, a porcentagem de reajuste e forneça o novo salário após o aumento por meio de uma função em "C".

## Resolução da situação-problema:

### Identificação dos Dados:

#### 1. Declaração da Função:

- Criar uma função "**calcularReajuste()**" que recebe como parâmetros o salário atual e o percentual de reajuste.

#### 2. Cálculo do Reajuste na Função:

- Implementar dentro da função o cálculo do reajuste utilizando a fórmula: "**reajuste = (salarioAtual \* percentualReajuste) / 100**".

#### 3. Retorno do novo salário:

- A função deve retornar o novo salário após o reajuste.

#### 4. Chamada da função no programa principal:

- No programa principal, solicitar ao usuário que informe o salário atual e o percentual de reajuste.
- Chamar a função calcularReajuste com os dados fornecidos pelo usuário.
- Exibir o resultado na tela.

#### 5. Implementação em Linguagem C (Exemplo):

Segue uma sugestão do algoritmo resolvido na linguagem de programação C.

# ALGORITMOS E TÉCNICAS DE PROGRAMAÇÃO

```
1 #include <stdio.h>
2
3 // Passo 1
4 float calcularReajuste(float salarioAtual, float percentualReajuste) {
5 // Passo 2
6 float reajuste = (salarioAtual * percentualReajuste) / 100;
7
8 // Passo 3
9 return salarioAtual + reajuste;
10 }
11
12 int main() {
13 float salarioAtual, percentualReajuste;
14
15 // Passo 4
16 printf("Informe o salário atual do colaborador: ");
17 scanf("%f", &salarioAtual);
18 printf("Informe o percentual de reajuste desejado: ");
19 scanf("%f", &percentualReajuste);
20
21 // Passo 5
22 float novoSalario = calcularReajuste(salarioAtual, percentualReajuste);
23
24 // Exibição dos Resultados
25 printf("O novo salário é R$ %.2f\n", novoSalario);
26
27 return 0;
28 }
```

Figura 7 | Algoritmo resolvido.

A implementação de uma função em C para calcular o reajuste salarial oferece uma abordagem modular e organizada, permitindo uma maior reutilização do código.

Essa estrutura funcional contribui para a legibilidade e manutenção do programa, refletindo a importância do uso de funções na programação. Ao aplicar esse conceito em um contexto prático, o estudante ganha experiência na criação de soluções mais estruturadas e eficientes.

## Saiba mais

Para saber mais sobre o desenvolvimento de algoritmos e suas pesquisas, acesse o portal da Scielo Brasil [scielo.br](http://scielo.br) e veja o artigo intitulado como: "[Validação de um Algoritmo de Inteligência](#)

# ALGORITMOS E TÉCNICAS DE PROGRAMAÇÃO

[Artificial para a Predição Diagnóstica de Doença Coronariana: Comparação com um Modelo Estatístico Tradicional](#). Boa leitura!

## Referências

CORMEN, T. **Algoritmos – Teoria e Prática**. 3.ed. Rio de Janeiro: LTC, 2022.

CORMEN, T. **Desmistificando Algoritmos**. Grupo GEN, 2013. E-book.

FERREIRA, Aurélio Buarque de Holanda. **Novo Aurélio Século XXI**: o dicionário da língua portuguesa. 3 ed. totalmente rev. e ampl. Rio de Janeiro: Nova Fronteira, 1999.

MANZANO, J.A. N G.; OLIVEIRA, J. F. de. **Estudo Dirigido de Algoritmos**. 15. ed. Editora Saraiva, 2012. E-book.

MENÉNDEZ, A. **Simplificando Algoritmos**. Grupo GEN, 2023. E-book.

PINHEIRO, F. de A. C. **Elementos de Programação em C**. Porto Alegre: Bookman, 2012.

SERPA, M. S.; RODRIGUES, T. N.; ALVES, Í. C.; et al. **Análise de Algoritmos**. Grupo A, 2021. E-book.

WAZLAWICK, R. S. **Introdução a Algoritmos e Programação com Python: Uma Abordagem Dirigida por Testes**. Rio de Janeiro: LTC, 2018.

## Aula 3

Recursividade

### Recursividade

# ALGORITMOS E TÉCNICAS DE PROGRAMAÇÃO



## Este conteúdo é um vídeo!

Para assistir este conteúdo é necessário que você acesse o AVA pelo computador ou pelo aplicativo. Você pode baixar os vídeos direto no aplicativo para assistir mesmo sem conexão à internet.

### Dica para você

Aproveite o acesso para baixar os slides do vídeo, isso pode deixar sua aprendizagem ainda mais completa.

Olá, estudante! Nesta videoaula, você verá sobre a recursividade, que em sua definição mais simples, é a capacidade de uma função chamar a si mesma. Essa técnica oferece uma abordagem elegante e poderosa para resolver problemas complexos, desmembrando-os em instâncias menores e mais gerenciáveis.

Nosso objetivo é que, ao entender os princípios fundamentais da recursividade, você seja capaz de aplicar essa técnica de maneira prática em seus futuros desafios de programação.

Vamos explorar um exemplo de recursividade simples durante a aula, como a implementação da sequência de Fibonacci ou o cálculo do fatorial de um número. Esses casos fornecem uma introdução clara aos conceitos básicos da recursividade, destacando a importância de condições de parada para evitar loops infinitos.

Além disso, vamos discutir como a recursividade pode ser incorporada em sistemas mais amplos. Portanto, ao entender como uma função pode chamar a si mesma de forma controlada e eficiente, você estará preparado para aplicar esse conhecimento em problemas do mundo real, onde a modularidade e a clareza são indispensáveis.

Os conhecimentos adquiridos nesta videoaula não apenas ampliarão sua compreensão da recursividade, mas também contribuirão para o fortalecimento de sua capacidade de pensamento crítico e resolução de problemas. Aproveite cada momento de aprendizado, participeativamente e esteja preparado para explorar os desafios que a recursividade pode resolver.

## Ponto de Partida

Olá, estudante! Nesta aula, nos aprofundaremos em conceitos de algoritmos e técnicas de programação, com um foco especial na fascinante área da recursividade.

A recursividade, em sua essência, é uma técnica que permite que uma função se chame a si própria. Este conceito, aparentemente simples, abre portas para soluções elegantes e poderosas na resolução de problemas complexos.

# ALGORITMOS E TÉCNICAS DE PROGRAMAÇÃO

Ao compreender os fundamentos da recursividade, você ganhará a habilidade de aplicar essa técnica de forma prática em desafios futuros de programação.

Para ilustrar essa ideia, vamos explorar um exemplo de recursividade simples, tal como a implementação da sequência de Fibonacci ou a resolução do cálculo fatorial. Estes exemplos oferecem uma entrada gradual aos princípios básicos da recursividade, ressaltando a importância de condições de parada para evitar iterações infinitas.

Além disso, vamos discutir como a recursividade pode ser incorporada de maneira inteligente em sistemas mais amplos. Ao compreender como uma função pode chamar a si mesma de maneira controlada e eficiente, você estará preparado para aplicar esse conhecimento em cenários do mundo real.

Diante disso, será apresentado o desenvolvimento de um algoritmo para realizar o abastecimento de caminhões que transportam combustível utilizando para isso a recursividade.

Os conhecimentos adquiridos nesta aula irão ampliar sua compreensão da recursividade, além de contribuir significativamente para fortalecer sua capacidade de pensamento crítico e resolução de problemas.

Vamos lá?

## Vamos Começar!

A recursividade é um conceito fundamental em ciência da computação e programação, referindo-se à capacidade de uma função chamar a si mesma durante sua execução. Essa abordagem oferece uma maneira elegante e poderosa de resolver problemas complexos, decompondo-os em subproblemas mais simples.

A definição de recursividade envolve a subdivisão de um problema em instâncias menores do mesmo problema, até atingir uma condição de parada bem definida. Esse método permite a construção de soluções mais claras e concisas para certos tipos de problemas, proporcionando eficiência e modularidade ao código.

Um exemplo simples de recursividade pode ser observado na implementação do cálculo fatorial de um número. Ao invés de usar um *loop* iterativo, uma função recursiva pode ser formulada, em que o cálculo do fatorial de um número é expresso em termos do cálculo do fatorial de números menores. Essa abordagem ilustra a característica autorreferencial da recursividade, em que a função invoca a si mesma para resolver partes menores do problema original.

Além disso, a recursividade desempenha um papel vital dentro de sistemas mais amplos, como algoritmos de ordenação, travessia de estruturas de dados e análise de árvores. Em estruturas de

# ALGORITMOS E TÉCNICAS DE PROGRAMAÇÃO

dados hierárquicas, como árvores binárias, a aplicação de funções recursivas facilita a manipulação eficiente dos elementos.

A recursividade permite lidar com a complexidade inerente a essas estruturas de maneira elegante, contribuindo para a legibilidade e manutenção do código.

No âmbito da programação, entender a definição de recursividade, explorar exemplos simples e compreender como aplicá-la eficazmente em sistemas mais complexos é essencial para desenvolvedores.

Este conjunto de habilidades não apenas amplia a gama de problemas que podem ser resolvidos de maneira eficiente, mas também promove uma compreensão mais profunda dos princípios fundamentais da computação. Ao dominar a recursividade, os programadores podem enfrentar desafios algorítmicos com confiança, construindo soluções robustas e eficazes.

## Definição de recursividade

A recursividade é caracterizado pela capacidade de uma função se chamar a si mesma durante sua execução. Essa abordagem é particularmente útil quando se lida com problemas que podem ser decompostos em subproblemas mais simples e semelhantes ao problema original.

A essência da recursividade reside na definição da função em termos de si mesma, o que permite a resolução de problemas complexos através de uma abordagem mais elegante e modular.

Assim, uma função recursiva possui dois componentes principais: o caso base e o caso recursivo. O caso base é a condição que determina quando a função deve parar de se chamar, evitando um ciclo infinito.

Já o caso recursivo é a parte da função que se chama novamente, geralmente resolvendo uma versão menor do problema original. Essa estrutura garante que a recursividade atinja uma solução ao reduzir gradualmente o problema até atingir o caso base.

Um exemplo clássico de recursividade é o cálculo do fatorial de um número. Se denotarmos o fatorial de um número  $n$  como  $n!$ , a função fatorial é definida como  $n!=n \times (n-1)!$ , com o caso base sendo  $0!=1$ .

Essa definição demonstra a natureza autorreferencial da recursividade, onde o cálculo do fatorial de um número é expresso em termos do cálculo do fatorial de um número menor.

A recursividade oferece uma maneira poderosa de resolver problemas complexos, proporcionando clareza e concisão ao código. No entanto, é importante usá-la com cuidado, pois uma implementação inadequada pode levar a problemas de desempenho e estouro de pilha.

# ALGORITMOS E TÉCNICAS DE PROGRAMAÇÃO

Compreender a recursividade é fundamental para desenvolvedores, pois ela não apenas amplia a gama de problemas que podem ser resolvidos de maneira eficiente, mas também contribui para uma compreensão mais profunda dos fundamentos da computação e da estrutura de algoritmos.

Dominar a recursividade é, portanto, uma habilidade valiosa para qualquer programador que busca construir soluções elegantes e eficazes para uma variedade de desafios computacionais.

## Exemplo de recursividade simples

Vamos considerar esse exemplo clássico de recursividade, que é o fatorial:

```
1 #include <stdio.h>
2
3 int factorial(int n) {
4 // Caso base
5 if (n == 0 || n == 1) {
6 return 1;
7 }
8 // Caso recursivo
9 else {
10 return n * factorial(n - 1);
11 }
12 }
13
14 int main() {
15 int numero = 5;
16 printf("O fatorial de %d é: %d\n", numero, factorial(numero));
17 return 0;
18 }
```

Figura 1 | Fatorial.

Neste exemplo, a função factorial é definida com um caso base que retorna 1 quando n é 0 ou 1, e um caso recursivo que calcula  $n \times (n-1)!$  O programa principal então imprime o resultado para o número 5. Essa implementação ilustra claramente a estrutura básica de uma função recursiva.

Entretanto, ao implementar recursividade, é importante estar ciente de potenciais erros, como estouro de pilha, que podem ocorrer se não houver um caso base adequado. Se esquecermos de

# ALGORITMOS E TÉCNICAS DE PROGRAMAÇÃO

incluir o caso base ou se ele for mal definido, a função pode entrar em um loop infinito, resultando em um estouro de pilha.

Uma alternativa interessante para evitar estouros de pilha é utilizar a recursividade de cauda. Neste caso, a chamada recursiva é a última operação realizada pela função, o que permite otimizações por parte do compilador. Aqui está uma versão de fatorial usando recursividade de cauda:

```
1 #include <stdio.h>
2
3 int fatorial_tail(int n, int resultado) {
4 // Caso base
5 if (n == 0 || n == 1) {
6 return resultado;
7 }
8 // Caso recursivo
9 else {
10 return fatorial_tail(n - 1, n * resultado);
11 }
12 }
13
14 int fatorial(int n) {
15 return fatorial_tail(n, 1);
16 }
17
18 int main() {
19 int numero = 5;
20 printf("O fatorial de %d é: %d\n", numero, fatorial(numero));
21 return 0;
22 }
23
```

Figura 2 | Outra alternativa.

Nesta versão, a função *fatorial\_tail* recebe um segundo argumento, que é o acumulador. Isso evita o estouro de pilha e permite uma implementação mais eficiente em termos de consumo de recursos. Essa variação ilustra como a recursividade pode ser adaptada para diferentes necessidades e situações, mostrando a versatilidade desse conceito na programação em C.

Vamos explorar outro exemplo clássico e simples de recursividade: a impressão dos primeiros n números naturais de forma decrescente em C. A função recursiva será responsável por imprimir

# ALGORITMOS E TÉCNICAS DE PROGRAMAÇÃO

os números de n até 1:

```
1 #include <stdio.h>
2
3 void imprimirNumerosDecrescentes(int n) {
4 // Caso base
5 if (n == 0) {
6 return;
7 }
8 // Caso recursivo
9 else {
10 printf("%d ", n);
11 imprimirNumerosDecrescentes(n - 1);
12 }
13 }
14
15 int main() {
16 int numero = 5;
17 printf("Os primeiros %d números naturais em ordem decrescente são: ", numero);
18 imprimirNumerosDecrescentes(numero);
19 printf("\n");
20
21 return 0;
22 }
23
```

Figura 3 | Primeiros números naturais.

A função *imprimirNumerosDecrescentes* tem um caso base definido quando n é igual a zero, em que a função simplesmente retorna sem fazer nada. O caso recursivo, por sua vez, imprime o número n e, em seguida, chama a função novamente com n-1.

Dessa forma, a função continua a chamar a si mesma até atingir o caso base.

Ao executar o programa, para n = 5, a saída seria:

- Os primeiros 5 números naturais em ordem decrescente são: 5 4 3 2 1.

Isso acontece porque a função recursiva imprime os números de 5 a 1, seguindo a lógica do caso base e caso recursivo. Este exemplo ilustra como a recursividade pode ser usada para criar soluções concisas e elegantes, decompondo o problema em partes menores e reutilizando a função para resolver subproblemas.

**Siga em Frente...**

# ALGORITMOS E TÉCNICAS DE PROGRAMAÇÃO

## A recursividade dentro de um sistema

Vamos estudar um caso de um problema real que foi resolvido com recursividade para entendermos melhor a importância dessa técnica de programação:

### Estudo de Caso

Resolvendo o Problema das Torres de Hanói com Recursividade

O problema das Torres de Hanói é um desafio clássico que pode ser resolvido eficientemente através da recursividade. A tarefa é mover uma pilha de discos de um pino de origem para um pino de destino, utilizando um pino intermediário, seguindo duas regras: apenas um disco pode ser movido por vez, e um disco maior nunca pode ficar sobre um disco menor.

### Descrição do Problema

Considere três pinos (A, B e C) e uma pilha de discos de tamanhos diferentes no pino A, empilhados do maior para o menor. O objetivo é mover todos os discos para o pino C, utilizando o pino B como intermediário, seguindo as regras mencionadas.

### Solução Recursiva

A abordagem recursiva para resolver esse problema é elegante. Seja moveTorre( $n$ , origem, destino, intermediario) uma função que move uma torre de  $n$  discos do pino de origem para o destino, utilizando o pino intermediário.

### Caso Base

Se  $n == 1$ , move o disco de origem para o destino e retorne.

### Caso Recursivo

- Mova  $n-1$  discos da origem para o pino intermediário usando o destino como pino auxiliar.
- Mova o disco restante da origem para o destino.
- Mova os  $n-1$  discos do pino intermediário para o destino usando a origem como pino auxiliar.

# ALGORITMOS E TÉCNICAS DE PROGRAMAÇÃO

```

1 #include <stdio.h>
2
3 void moveTorre(int n, char origem, char destino, char intermediario) {
4 if (n == 1) {
5 printf("Mova disco de %c para %c\n", origem, destino);
6 return;
7 }
8 moveTorre(n-1, origem, intermediario, destino);
9 printf("Mova disco de %c para %c\n", origem, destino);
10 moveTorre(n-1, intermediario, destino, origem);
11 }
12
13 int main() {
14 moveTorre(3, 'A', 'C', 'B');
15 return 0;
16 }
17

```

Figura 4 | Torre de Hanói.

- A função *moveTorre* é chamada inicialmente com  $n$  discos do pino A para o pino C, utilizando o pino B como intermediário.
- O caso base é quando  $n == 1$ , onde o disco é movido diretamente do pino de origem para o destino.
- No caso recursivo, a função é chamada duas vezes, primeiro movendo  $n-1$  discos para o pino intermediário e, em seguida, movendo o disco restante para o destino.
- Isso continua até que o caso base seja alcançado, movendo cada disco da origem para o destino, respeitando as regras do problema.

## Conclusão

A recursividade, neste caso, oferece uma solução concisa e eficiente para o problema das Torres de Hanói, dividindo-o em subproblemas menores. Essa abordagem é uma aplicação prática e clássica da recursividade na resolução de um problema do mundo real.

A utilização da recursividade no exemplo das Torres de Hanói destaca a sua importância na programação por diversas razões. Primeiramente, a recursividade permite expressar soluções de forma concisa e clara, facilitando a compreensão do código.

Além disso, essa abordagem favorece a divisão natural de problemas complexos em subproblemas mais simples, tornando a resolução mais intuitiva.

# ALGORITMOS E TÉCNICAS DE PROGRAMAÇÃO

A reutilização de código é incentivada pela recursividade, já que a mesma função pode ser chamada com diferentes argumentos para abordar diferentes instâncias do problema. Essa característica promove uma abordagem modular e eficiente na construção de soluções.

No contexto de estruturas de dados recursivas, como árvores ou listas encadeadas, a recursividade é uma escolha natural para percorrer ou manipular essas estruturas, simplificando a implementação.

Algoritmos de divisão e conquista, como o utilizado no exemplo das Torres de Hanói, muitas vezes são mais eficientes e elegantes quando implementados de forma recursiva. Essa abordagem permite a expressão direta desses algoritmos, resultando em soluções mais eficientes.

Além disso, a recursividade contribui para a facilidade na manutenção e entendimento do código, especialmente em problemas complexos, devido à sua clara divisão em casos base e casos recursivos.

A recursividade oferece flexibilidade na resolução de problemas, adaptando-se a diferentes contextos e tipos de desafios de maneira eficaz. Isso a torna uma ferramenta valiosa para programadores enfrentarem uma variedade de problemas computacionais.

Também, a expressividade e elegância proporcionadas pela recursividade contribuem para a legibilidade e compreensão do código-fonte, facilitando a colaboração entre desenvolvedores.

Ou seja, a recursividade é uma técnica valiosa na programação devido à sua capacidade de simplificar problemas complexos, promover clareza no código e oferecer soluções elegantes e eficientes.

A escolha entre abordagens recursivas e iterativas depende do contexto do problema, mas a recursividade continua sendo uma ferramenta amplamente utilizada e essencial no arsenal de programadores.

## Vamos Exercitar?

### Descrição da situação-problema:

Imagine que você é responsável por desenvolver um programa para uma empresa de logística que gerencia o abastecimento de sua frota de caminhões de combustível. A empresa precisa de uma solução eficiente para calcular os custos totais de abastecimento, levando em consideração diferentes tipos de combustíveis e os volumes necessários para cada veículo.

### Enunciado:

# ALGORITMOS E TÉCNICAS DE PROGRAMAÇÃO

A empresa possui uma variedade de caminhões, cada um com capacidades distintas de tanque e requerimentos específicos de combustível. A necessidade é criar um algoritmo que, utilizando recursividade, calcule os custos totais de abastecimento para todos os veículos, considerando a variedade de combustíveis disponíveis.

**Resolução da situação-problema:**

**Definição da Função Recursiva:**

```

2 float calcularCustoTotal(int indiceCaminhao, float custoTotalAnterior) {
3 if (indiceCaminhao < totalCaminhos) {
4 // Lógica para calcular custo do abastecimento do caminhão atual
5 custoTotalAnterior += calcularCustoAbastecimento(caminhos[indiceCaminhao]);
6 // Chamada recursiva para o próximo caminhão
7 return calcularCustoTotal(indiceCaminhao + 1, custoTotalAnterior);
8 } else {
9 // Caso base: todos os caminhões foram abastecidos
10 return custoTotalAnterior;
11 }
12 }
```

Figura 5 | Função recursiva.

**Chamada da Função:**

```

25 float custoTotal = calcularCustoTotal(0, 0.0);
26
```

Figura 6 | Chamada da função.

**Exibição do Resultado:**

```
34 printf("O custo total de abastecimento para a frota é: R$ %.2f", custoTotal);
```

Figura 7 | Resultado.

**Conclusão**

A utilização da recursividade nesse contexto de controle de abastecimento de caminhões proporciona uma implementação modular e elegante. O algoritmo é capaz de lidar com diferentes quantidades e tipos de caminhões de maneira eficiente, facilitando a manutenção e expansão do sistema.

# ALGORITMOS E TÉCNICAS DE PROGRAMAÇÃO

Contudo, é fundamental considerar o desempenho em situações de frota muito extensa, onde abordagens iterativas poderiam ser mais eficazes. Em geral, a recursividade destaca-se pela clareza e simplicidade, sendo uma escolha valiosa em muitos contextos de programação.

## Saiba mais

Para saber mais sobre recursividade e funções recursivas, acesse em sua Biblioteca Virtual o livro [\*Introdução a Algoritmos e Programação com Python: Uma Abordagem Dirigida por Testes\*](#), Capítulo 7.9 - Recursividade.

## Referências

CORMEN, T. **Algoritmos – Teoria e Prática**. 3.ed. Rio de Janeiro: LTC, 2022.

CORMEN, T. **Desmistificando Algoritmos**. Grupo GEN, 2013. E-book.

FERREIRA, Aurélio Buarque de Holanda. **Novo Aurélio Século XXI**: o dicionário da língua portuguesa. 3 ed. totalmente rev. e ampl. Rio de Janeiro: Nova Fronteira, 1999.

MANZANO, J.A. N G.; OLIVEIRA, J. F. de. **Estudo Dirigido de Algoritmos**. 15. ed. Editora Saraiva, 2012. E-book.

MENÉNDEZ, A. **Simplificando Algoritmos**. Grupo GEN, 2023. E-book.

PINHEIRO, F. de A. C. **Elementos de Programação em C**. Porto Alegre: Bookman, 2012.

SERPA, M. S.; RODRIGUES, T. N.; ALVES, I. C.; *et al.* **Análise de Algoritmos**. Grupo A, 2021. E-book.

WAZLAWICK, R. S. **Introdução a Algoritmos e Programação com Python: Uma Abordagem Dirigida por Testes**. Rio de Janeiro: LTC, 2018.

## Aula 4

Funções Recursivas

### Funções recursivas

# ALGORITMOS E TÉCNICAS DE PROGRAMAÇÃO



## Este conteúdo é um vídeo!

Para assistir este conteúdo é necessário que você acesse o AVA pelo computador ou pelo aplicativo. Você pode baixar os vídeos direto no aplicativo para assistir mesmo sem conexão à internet.

### Dica para você

Aproveite o acesso para baixar os slides do vídeo, isso pode deixar sua aprendizagem ainda mais completa.

Olá, estudante! Nesta videoaula abordaremos assuntos relativos às funções recursivas, como definição de funções recursivas, funções recursivas em cauda e exemplos de funções recursivas em um sistema.

Para que você comprehenda os conteúdos da melhor forma possível, vamos trazer exemplos em que esses conteúdos podem ser aplicados na prática, tornando seu aprendizado mais significativo.

Os conhecimentos adquiridos aqui não apenas expandem a sua compreensão do assunto, mas também fortalecem a sua capacidade de pensamento crítico e resolução de problemas. Aproveite cada oportunidade e cada leitura para explorar os conceitos apresentados, participando ativamente na construção do seu conhecimento.

Vamos lá!

## Ponto de Partida

Olá, estudante. Nesta aula, vamos explorar conteúdos referentes a algoritmos e técnicas de programação e teremos como foco o estudo de funções recursivas.

Dominar esse conceito permitirá a você criar programas mais dinâmicos e adaptáveis, capazes de responder de maneira inteligente a diferentes cenários. Além disso, você vai compreender a capacidade dessa abordagem em expressar soluções de forma elegante e eficiente para problemas que possuem uma estrutura recursiva inerente.

Por fim, o estudo da recursividade, é fundamental para lidar com situações algorítmicas complexas que precisam desse recurso. Dominar essa técnica permitirá você a desenvolver algoritmos com maior performance, capazes de lidar com uma ampla gama de situações do mundo real.

Diante disso, será apresentado o desenvolvimento de um algoritmo para cálculo do fatorial de um número escolhido pelo usuário.

Convidamos você a iniciar a construção de uma base sólida para sua jornada no universo da programação. Vamos começar?

# ALGORITMOS E TÉCNICAS DE PROGRAMAÇÃO

Vamos Começar!

## Definição de funções recursivas

As funções recursivas são uma evolução ao conteúdo de funções. Antes de compreender o que são funções recursiva é preciso estudar o conceito de recursividade e até mesmo funções no seu aspecto simples, ou seja, procedimento, funções, funções com ou sem parâmetros, funções com ou sem retorno etc.

Um exemplo claro de recursividade é quando temos uma imagem refletida dentro dela mesma várias vezes dando um efeito de infinito. Observe a Figura 1.



Figura 1 | Imagem recursiva.

A recursividade dentro da programação se caracteriza quando em uma determinada linha de código de uma função, existe a chamada para a mesma função, ou seja, ela está executando a si própria.

Na Figura 1, podemos visualizar o mesmo efeito. É como se dentro da imagem tivéssemos novamente a mesma imagem, que dentro dela terá novamente a imagem, e assim, sucessivamente.

# ALGORITMOS E TÉCNICAS DE PROGRAMAÇÃO

Como seria a programação, o código fictício de uma função na linguagem C, que apresentasse o que estamos vendo na imagem da Figura 1?

```
1 #include <stdio.h>
2
3 ∵ int imagem(char imagem) {
4 // Caso Base
5 }
6
7
8 ∵ int main() {
9
10 // Exemplo de uso da função
11 imagem("Carregar a imagem do dispositivo");
12
13 return 0;
14 }
```

Figura 2 | Código da imagem recursiva.

Observe, na Figura 2, que dentro da função principal *main()* a função **imagem()** é chamada novamente. Este processo vai criar o efeito que é exibido na Figura 1.

Diante do exposto, funções recursivas são aquelas que invocam a si mesmas durante sua execução. Na linguagem C, essa técnica é implementada quando uma função chama a si própria para resolver instâncias menores do mesmo problema.

Ainda, uma função recursiva consiste em dois componentes essenciais: o caso base e o caso recursivo. O caso base define a condição que encerra as chamadas recursivas, como se fosse um laço de repetição, a condição de parada ou de saída do laço. Enquanto o caso recursivo representa a chamada à própria função para resolver instâncias menores do problema.

Elas desempenham um papel fundamental na programação, especialmente na linguagem C. Elas oferecem uma abordagem poderosa e elegante para resolver problemas que podem ser decompostos em subproblemas menores.

# ALGORITMOS E TÉCNICAS DE PROGRAMAÇÃO

A estrutura básica de uma função recursiva envolve a definição do caso base e a chamada recursiva. Por exemplo, considere o cálculo do fatorial:

```
1 ✓ int factorial(int n) {
2 // Caso Base
3 if (n == 0 || n == 1) {
4 return 1;
5 }
6 // Caso Recursivo
7 else {
8 return n * factorial(n - 1);
9 }
10 }
```

Figura 3 | Função recursiva.

## Vantagens e desvantagens

As funções recursivas podem simplificar a implementação de algoritmos e tornar o código mais legível. No entanto, é essencial gerenciar cuidadosamente as chamadas recursivas para evitar estouro de pilha e garantir eficiência.

Para ilustrar vamos abordar a sequência de Fibonacci. A série de Fibonacci é uma sequência em que cada número é a soma dos dois anteriores (0, 1, 1, 2, 3, 5, 8, 13, ...). Observe a Figura 4.

# ALGORITMOS E TÉCNICAS DE PROGRAMAÇÃO

```
1 #include <stdio.h>
2
3 // Função recursiva para calcular o termo de Fibonacci
4 int fibonacci(int n) {
5 if (n <= 1) {
6 return n;
7 } else {
8 return fibonacci(n - 1) + fibonacci(n - 2);
9 }
10 }
11
12 int main() {
13 int termo;
14
15 // Solicitar ao usuário o termo desejado
16 printf("Digite o termo de Fibonacci desejado: ");
17 scanf("%d", &termo);
18
19 // Calcular e exibir o termo de Fibonacci
20 printf("O termo de Fibonacci de %d é %d\n", termo, fibonacci(termo));
21
22 return 0;
23 }
24
```

Figura 4 | Série de Fibonacci.

Este programa solicitará ao usuário um termo desejado da sequência de Fibonacci e calculará o valor usando a função Fibonacci. É importante lembrarmos de que a abordagem recursiva pode se tornar ineficiente para valores grandes devido ao grande número de chamadas de função. Para otimizar isso, podemos considerar abordagens iterativas ou memorização.

## Funções recursivas em cauda

Além das funções recursivas já abordadas no tópico anterior, temos outras variações, entre elas estão as funções recursivas em cauda.

Funções recursivas em cauda ocorrem quando a chamada recursiva é a última ação executada dentro da função.

Diferentemente das funções recursivas tradicionais, em que as chamadas recursivas podem resultar em operações adicionais após o retorno, as funções em cauda são estruturadas de

# ALGORITMOS E TÉCNICAS DE PROGRAMAÇÃO

maneira a otimizar essa operação.

Entre as vantagens da utilização dessas operações, estão:

- **Eficiência de espaço:** a otimização de cauda permite que o compilador utilize uma quantidade constante de espaço na pilha, independentemente do número de chamadas recursivas.
- **Desempenho otimizado:** a execução em cauda pode ser mais eficiente, pois evita a acumulação de chamadas na pilha.

As funções recursivas em cauda proporcionam uma abordagem eficiente para resolver problemas recursivos, especialmente em linguagens que suportam otimizações específicas. Ao entender e aplicar esse conceito, os programadores podem escrever código mais eficiente e econômico em termos de recursos, promovendo boas práticas de programação.

No entanto, é importante lembrar que nem todas as linguagens ou compiladores oferecem suporte total a otimizações de cauda, sendo essencial verificar a documentação específica da linguagem ou compilador utilizado.

Um exemplo prático da utilização da recursividade em cauda por meio da linguagem de programação C pode ser visualizado na Figura 5.

# ALGORITMOS E TÉCNICAS DE PROGRAMAÇÃO

```
1 #include <stdio.h>
2
3 // Função recursiva em cauda para cálculo de factorial
4 int factorial_tail(int n, int resultado) {
5 if (n == 0) {
6 return resultado;
7 } else {
8 return factorial_tail(n - 1, n * resultado);
9 }
10 }
11
12 // Função wrapper para chamada inicial
13 int factorial(int n) {
14 return factorial_tail(n, 1);
15 }
16
17 int main() {
18 int numero = 5;
19 printf("O factorial de %d é %d\n", numero, factorial(numero));
20 return 0;
21 }
22
```

Figura 5 | Exemplo de recursividade com cauda.

No exemplo da Figura 5, a função `factorial_tail()` é recursiva em cauda, acumulando o resultado durante as chamadas. Dentro da função `main()`, o comando `printf` exibe o resultado do valor do número periodicamente.

As funções recursivas em cauda proporcionam uma abordagem eficiente para resolver problemas recursivos, especialmente em linguagens que suportam otimizações específicas. Ao entender e aplicar esse conceito, os programadores podem escrever um código mais eficiente e econômico em termos de recursos, promovendo boas práticas de programação.

No entanto, é importante lembrar que nem todas as linguagens ou compiladores oferecem suporte total a otimizações de cauda, sendo essencial verificar a documentação específica da linguagem ou compilador utilizado.

**Siga em Frente...**

# ALGORITMOS E TÉCNICAS DE PROGRAMAÇÃO

## Exemplos de funções recursivas em um sistema

A utilização de funções recursivas em sistemas de software é uma prática que se destaca pela sua elegância e eficiência. A recursividade, como técnica de resolução de problemas, proporciona uma abordagem estruturada e modular, permitindo a decomposição de tarefas complexas em componentes menores e mais gerenciáveis.

No contexto de sistemas, as funções recursivas emergem como ferramentas poderosas, oferecendo soluções flexíveis e adaptáveis a uma variedade de desafios.

Ao explorar exemplos práticos de funções recursivas, é possível identificar áreas em que essa abordagem se destaca. Desde a manipulação de estruturas de dados até a resolução de problemas específicos do domínio, a recursividade proporciona soluções que simplificam o código e refletem a natureza intrínseca de muitos problemas do mundo real.

Neste contexto, este tópico aprofundará os exemplos de funções recursivas em sistemas, destacando casos de uso comuns e demonstrando como essa abordagem pode contribuir para a criação de software robusto e eficiente.

A compreensão desses exemplos enriquece o repertório do desenvolvedor, além de, oferecer insights valiosos sobre a aplicação estratégica da recursividade para desafios específicos enfrentados na construção e manutenção de sistemas de software. Vamos explorar, portanto, como a elegância da recursividade pode ser efetivamente traduzida em soluções práticas que elevam a qualidade e a modularidade dos sistemas.

As funções recursivas e ou recursivas em caudas são utilizadas em sistemas em diversos cenários, podemos citar:

### 1. Cálculos matemáticos iterativos:

- Funções recursivas em cauda são eficientes para cálculos matemáticos iterativos, como fatoriais, combinações e permutações. Inclusive alguns desses são frequentemente utilizados academicamente para estudo e pesquisa.
- A otimização de cauda elimina a necessidade de armazenar várias chamadas na pilha, economizando espaço e melhorando o desempenho.

### 2. Trabalho com estrutura de dados:

- Ao manipular estruturas de dados recursivamente, como listas ou árvores, funções em cauda podem ser aplicadas para percorrer e processar os elementos de forma eficiente.
- A otimização de cauda evita o estouro da pilha de chamadas, tornando a solução mais escalável.

# ALGORITMOS E TÉCNICAS DE PROGRAMAÇÃO

## 3. Análise e transformação de *Strings*:

- Algoritmos que envolvem análise e transformação de *strings*, como inversão, palíndromos e padrões de busca, podem se beneficiar de funções recursivas em cauda.
- A execução eficiente permite processar grandes *strings* sem incorrer em custos adicionais de memória.

## 4. Resolução de Problemas de árvore e grafos:

- Em problemas que envolvem estruturas de árvore ou grafo, como travessias (DFS - *Depth-First Search*), funções recursivas em cauda podem ser aplicadas para explorar e processar nós.
- A otimização de cauda contribui para a eficiência em termos de espaço, especialmente em grafos profundos.

## 5. Algoritmos de *backtracking*:

- Algoritmos de *backtracking*, comuns em problemas de combinação, permutação e resolução de quebra-cabeças, podem ser implementados eficientemente com funções recursivas em cauda.
- A eliminação da sobrecarga na pilha permite uma busca mais profunda e eficiente.

Da mesma maneira que são utilizadas, também existe uma motivação, ou seja, o porquê funções recursivas em cauda são aliadas. Vamos aos motivos:

### 1. Eficiência de espaço:

- A otimização de cauda reduz o uso de espaço na pilha, permitindo o processamento eficiente de grandes conjuntos de dados sem o risco de estouro de pilha.

### 2. Desempenho aprimorado:

- Em comparação com funções recursivas tradicionais, as funções em cauda proporcionam desempenho aprimorado, pois evitam o acúmulo desnecessário de chamadas na pilha.

### 3. Escalabilidade:

- Em cenários em que a escalabilidade é crucial, como processamento de grandes volumes de dados, funções recursivas em cauda oferecem uma solução mais robusta e eficiente.

### 4. Legibilidade do código:

- Em muitos casos, funções recursivas em cauda resultam em códigos mais legíveis e elegantes, facilitando a compreensão e manutenção do código fonte.

# ALGORITMOS E TÉCNICAS DE PROGRAMAÇÃO

## 5. Aplicações em linguagens funcionais:

- Linguagens funcionais, que favorecem o paradigma de programação funcional, frequentemente incentivam o uso de funções recursivas em cauda devido às suas características e otimizações específicas.

## 6. Prevenção de Estouro de Pilha:

- O uso de funções recursivas em cauda é uma estratégia eficaz para prevenir estouros de pilha em casos de profundidade excessiva de chamadas recursivas.

Em resumo, as funções recursivas em cauda são aliadas em cenários que exigem eficiência em termos de espaço, desempenho otimizado e escalabilidade, tornando-as uma escolha valiosa em diversos contextos de programação.

## Vamos Exercitar?

### Descrição da situação-problema:

Imagine que você está desenvolvendo um programa para uma empresa que precisa calcular o fatorial de um número fornecido pelo usuário. O cálculo do fatorial é uma operação essencial em diversas aplicações, como análise estatística, modelagem matemática e processamento de dados.

No contexto dessa empresa, a eficiência na implementação desse cálculo é crucial, pois o programa será parte integrante de um sistema maior.

### Enunciado:

A empresa solicita que o programa seja capaz de calcular o fatorial de um número de forma rápida e eficiente, garantindo a correta execução mesmo para valores grandes. A escolha por uma implementação recursiva se dá pela clareza do código e pela facilidade de manutenção. A ideia é criar uma solução que seja elegante, fácil de entender e, ao mesmo tempo, eficaz.

### Resolução da situação-problema:

#### Definição da Função Recursiva:

# ALGORITMOS E TÉCNICAS DE PROGRAMAÇÃO

```
2 int calcularFatorial(int n) {
3 if (n == 0 || n == 1) {
4 return 1;
5 } else {
6 return n * calcularFatorial(n - 1);
7 }
8 }
```

Figura 6 | Função recursiva.

## Chamada da Função:

```
2 int calcularFatorial(int n) {
3 if (n == 0 || n == 1) {
4 return 1;
5 } else {
6 return n * calcularFatorial(n - 1);
7 }
8 }
9
10 int main(){
11
12 int numero = 5; // Pode ser qualquer número fornecido pelo usuário
13 int resultado = calcularFatorial(numero);
14
15 return 0;
16 }
```

Figura 7 | Chamada da função.

## Exibição do Resultado:

```
14
15 printf("O fatorial de %d é: %d", numero, resultado);
16
```

Figura 8 | Resultado.

# ALGORITMOS E TÉCNICAS DE PROGRAMAÇÃO

Este código implementa a lógica descrita na situação problema utilizando desvios condicionais (*if*) para lidar com diferentes situações durante a viagem à Lua. Vale ressaltar que os comentários indicam os passos correspondentes da solução.

## Conclusão

A utilização da recursividade para calcular o factorial apresenta-se como uma abordagem elegante e eficaz. O código torna-se mais legível e modular, facilitando a compreensão e manutenção.

No entanto, é essencial considerar que, em casos de números muito grandes, a recursividade pode levar a estouro de pilha, e nesses cenários, abordagens iterativas podem ser mais apropriadas.

Em geral, a escolha entre abordagens deve ser feita considerando as características específicas de cada situação. A implementação recursiva destaca-se pela clareza e simplicidade, fatores valiosos em muitos contextos de programação.

## Saiba mais

Para saber mais sobre recursividade e funções recursivas, acesse em sua Biblioteca Virtual o livro [Introdução a Programação Python: aprenda de forma rápida](#), Capítulo 4 – Funções e Módulo, item Funções recursivas.

## Referências

CORMEN, T. **Algoritmos – Teoria e Prática**. 3.ed. Rio de Janeiro: LTC, 2022.

CORMEN, T. **Desmistificando Algoritmos**. Grupo GEN, 2013. E-book.

FERREIRA, Aurélio Buarque de Holanda. **Novo Aurélio Século XXI**: o dicionário da língua portuguesa. 3 ed. totalmente rev. e ampl. Rio de Janeiro: Nova Fronteira, 1999.

MANZANO, J.A. N G.; OLIVEIRA, J. F. de. **Estudo Dirigido de Algoritmos**. 15. ed. Editora Saraiva, 2012. E-book.

MENÉNDEZ, A. **Simplificando Algoritmos**. Grupo GEN, 2023. E-book.

PINHEIRO, F. de A. C. **Elementos de Programação em C**. Porto Alegre: Bookman, 2012.

SERPA, M. S.; RODRIGUES, T. N.; ALVES, I. C.; et al. **Análise de Algoritmos**. Grupo A, 2021. E-book.

# ALGORITMOS E TÉCNICAS DE PROGRAMAÇÃO

WAZLAWICK, R. S. **Introdução a Algoritmos e Programação com Python: Uma Abordagem Dirigida por Testes.** Rio de Janeiro: LTC, 2018.

## Aula 5

Encerramento da Unidade

### Videoaula de Encerramento

#### Este conteúdo é um vídeo!



Para assistir este conteúdo é necessário que você acesse o AVA pelo computador ou pelo aplicativo. Você pode baixar os vídeos direto no aplicativo para assistir mesmo sem conexão à internet.

#### Dica para você

Aproveite o acesso para baixar os slides do vídeo, isso pode deixar sua aprendizagem ainda mais completa.

Olá, estudante! É um prazer tê-lo em nossa videoaula dedicada ao tema "Funções e recursividade". Neste encontro, exploraremos os conceitos e características essenciais presentes nesse específico conteúdo de algoritmos.

Durante a apresentação, aprofundaremos nosso entendimento nas chamadas "funções recursivas" e nas possibilidades dinâmicas de construção de algoritmos com a utilização dos recursos que as funções recursivas oferecem.

Portanto, não perca esta oportunidade! Inicie a aula, aproveite cada momento e vamos juntos neste aprendizado, que é fundamental para se destacar no universo dos algoritmos.

### Ponto de Chegada

Olá, estudante! Para desenvolver a competência desta Unidade, que é "Conhecer e aplicar os conceitos de recursividade e funções", devemos, primeiramente, conhecer o princípio da divisão de tarefas dentro de um algoritmo, que nada mais é que a divisão de partes do código em diversas partes menores a fim de resolver um problema maior dentro da proposta do algoritmo.

Compreender a lógica existente nas funções recursivas se faz necessário para a construção de programas que atendam às necessidades em diferentes contextos.

# ALGORITMOS E TÉCNICAS DE PROGRAMAÇÃO

Durante a abordagem do conteúdo, o estudante aprenderá as principais características das funções e funções recursivas, para realizar escolhas lógicas em seus algoritmos. Vamos compreender também como implementar esses recursos no desenvolvimento de soluções algorítmicas.

A aptidão para reconhecer e entender tais estruturas concede ao estudante uma base sólida na abordagem de problemas complexos, visto que muitas situações do cotidiano exigem algoritmos capazes de operar com exímia precisão além de uma alta performance. Essa habilidade é fundamental para quem almeja sobressair-se na programação e na resolução de desafios computacionais.

Em resumo, a identificação e compreensão dessas estruturas são fundamentais para construção de algoritmos sofisticados e eficientes. Essas competências capacitam o estudante a enfrentar desafios práticos na programação, preparando-o para abordar questões complexas em diversas áreas da computação.

O aprimoramento dessa habilidade representa, portanto, um significativo avanço na formação de um programador talentoso e inovador.

## É Hora de Praticar!

Suponha que você seja responsável por desenvolver uma função em linguagem C para calcular a média final da turma de um Curso de Programação. A turma possui um número variável de alunos, e cada aluno tem um conjunto de notas associadas às atividades do curso.

O desafio é desenvolver a solução utilizando o princípio da recursividade e funções recursivas. A sugestão de passos para a criação de uma solução dessa funcionalidade está descrita a seguir no tópico **Gabarito**.

### Número de Alunos:

- A função deve receber como parâmetro o número total de alunos na turma.

### Entrada de Notas:

- Cada aluno tem um conjunto de notas referentes às atividades do curso (provas, trabalhos, etc.).
- A função deve permitir a entrada das notas de cada aluno.

### Cálculo da Média:

- A média deve ser calculada de forma recursiva, somando as notas de todos os alunos e dividindo pelo número total de alunos.

### Exceções:

# ALGORITMOS E TÉCNICAS DE PROGRAMAÇÃO

- A função deve tratar possíveis exceções, como notas fora do intervalo esperado ou número inválido de alunos.

## Retorno:

- A função deve retornar a média final da turma.
- Ao explorar os conceitos de procedimento e funções na programação, reflita sobre como essas estruturas influenciam na organização e modularidade do código. Pergunte-se: de que maneira a utilização de procedimentos e funções facilita a compreensão e manutenção do código?
- Considere também situações práticas em que a escolha entre procedimentos e funções pode impactar a eficiência e a legibilidade do programa.
- Ao abordar o tema das funções recursivas, pense sobre a elegância e complexidade que esse conceito traz para a solução de problemas. Questione-se sobre os benefícios e desafios de implementar algoritmos recursivos.
- Como a recursividade pode tornar o código mais conciso, mas também mais difícil de entender? Explore exemplos em que funções recursivas são particularmente eficazes e casos em que podem ser menos vantajosas.
- Ao refletir sobre procedimentos, funções e funções recursivas, percebemos que esses conceitos são pilares fundamentais na construção de algoritmos eficientes e compreensíveis. A modularidade proporcionada por procedimentos e funções simplifica a manutenção do código, enquanto a recursividade oferece elegância em soluções complexas.

Essas reflexões reforçam a importância do entendimento profundo desses temas para a formação de programadores capazes e criativos. O aprendizado constante e a aplicação prática desses conceitos são passos essenciais rumo à maestria na arte da programação.

## Sugestão de solução

Para criar essa função recursiva, siga os seguintes passos:

### Passo 1: Definição de Parâmetros:

- Realizar uma reunião com a equipe de desenvolvimento de software e com os *stakeholders* para entender completamente os requisitos e expectativas em relação à funcionalidade de investimentos simples.

### Passo 2: Condição de Parada:

- Avaliar a viabilidade técnica e financeira da implementação da funcionalidade de investimentos simples no aplicativo do banco.
- Considerar os requisitos de segurança e conformidade regulatória para garantir a proteção dos dados e transações dos usuários.

### Passo 3: Soma Recursiva:

# ALGORITMOS E TÉCNICAS DE PROGRAMAÇÃO

- Criar um design detalhado da interface do usuário para a funcionalidade de investimentos simples, garantindo que seja intuitiva e fácil de usar.
- Definir os fluxos de navegação e interação do usuário durante o processo de investimento.

## Passo 4: Cálculo da Média:

- Implementar a funcionalidade de investimentos simples no aplicativo, seguindo as melhores práticas de desenvolvimento de software.
- Realizar testes rigorosos para garantir que a funcionalidade esteja livre de erros e atenda às expectativas dos usuários.

## Passo 5: Tratamento de Exceções:

- Integrar a funcionalidade de investimentos simples com os sistemas existentes do banco, como o sistema de contas dos usuários e o sistema de transações financeiras.

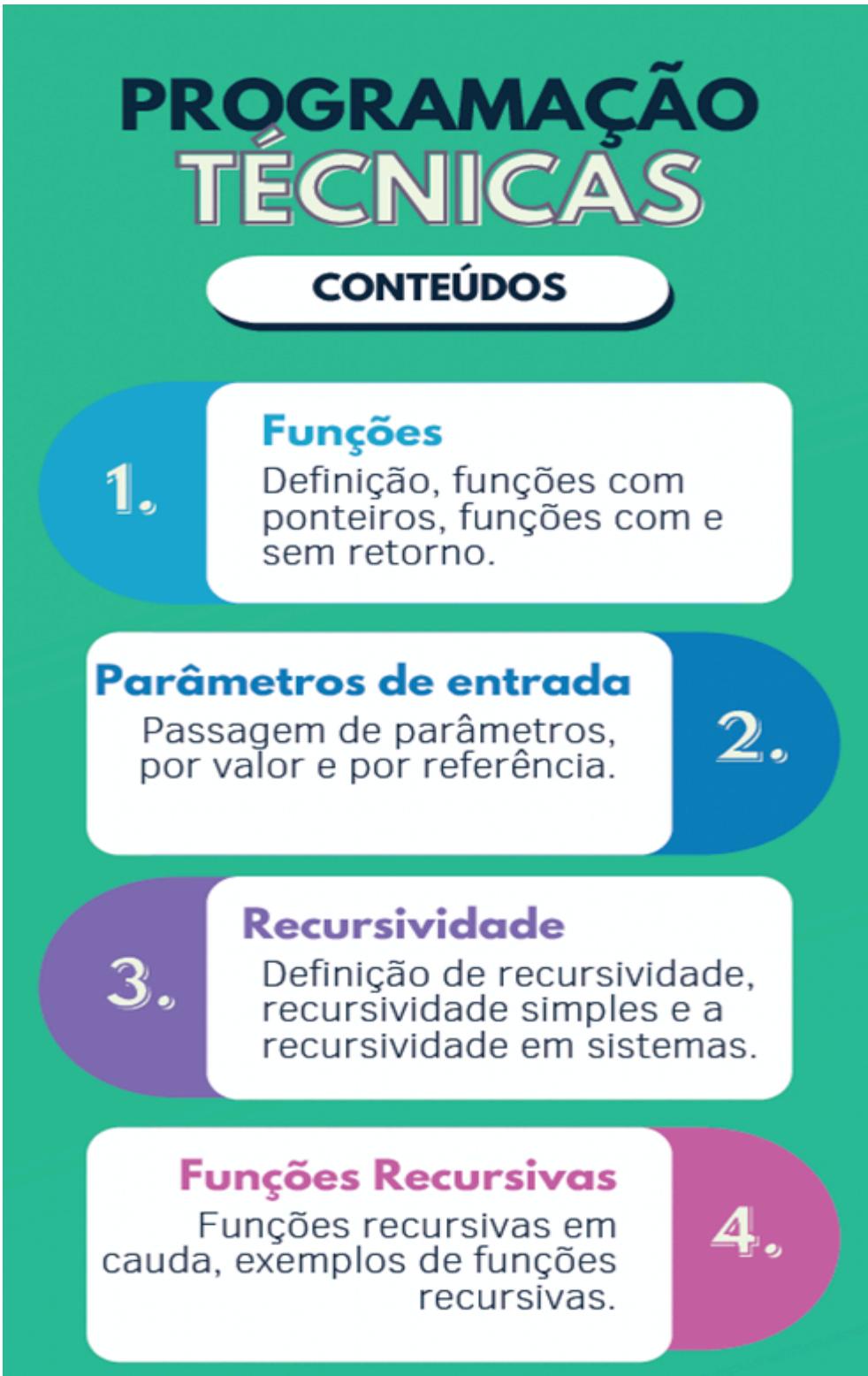
## Conclusão

Esse estudo de caso visa criar uma função que, de maneira eficiente e modular, calcula a média final da turma, permitindo flexibilidade e manutenção do código. A compreensão profunda desses passos é de vital importância para o desenvolvimento de funções recursivas e a utilização dos princípios da recursividade.

Nesta linha do tempo, apresentamos resumidamente os principais conceitos relacionados às funções e recursividade e como eles são utilizados na programação. Todos esses, são tópicos fundamentais para a disciplina de Algoritmos e Técnicas de Programação.

Confira a seguir:

# ALGORITMOS E TÉCNICAS DE PROGRAMAÇÃO



**PROGRAMAÇÃO TÉCNICAS**

**CONTEÚDOS**

- 1.** **Funções**  
Definição, funções com ponteiros, funções com e sem retorno.
- 2.** **Parâmetros de entrada**  
Passagem de parâmetros, por valor e por referência.
- 3.** **Recursividade**  
Definição de recursividade, recursividade simples e a recursividade em sistemas.
- 4.** **Funções Recursivas**  
Funções recursivas em cauda, exemplos de funções recursivas.

Figura | Técnicas de programação

CORMEN, T. **Algoritmos** – Teoria e Prática. 3.ed. Rio de Janeiro: LTC, 2022.

# ALGORITMOS E TÉCNICAS DE PROGRAMAÇÃO

CORMEN, T. **Desmistificando Algoritmos.** [S. I.]: Grupo GEN, 2013. E-book.

FERREIRA, A B H. **Novo Aurélio Século XXI:** o dicionário da língua portuguesa. 3 ed. Rio de Janeiro: Nova Fronteira, 1999.

MANZANO, J. A. N. G.; OLIVEIRA, J. F. **Estudo Dirigido de Algoritmos.** 15. ed. [S. I.]: Editora Saraiva, 2012.

MENÉNDEZ, A. **Simplificando Algoritmos.** [S. I.]: Grupo GEN, 2023.

PINHEIRO, F. A. C. **Elementos de Programação em C.** Porto Alegre: Bookman, 2012.

SERPA, M. S.; et al. **Análise de Algoritmos.** [S. I.]: Grupo A, 2021.

WAZLAWICK, R. S. **Introdução a Algoritmos e Programação com Python: Uma Abordagem Dirigida por Testes.** Rio de Janeiro: LTC, 2018.