

**ALGORITMOS E LÓGICA DE PROGAMAÇÃO**

**Aluno: Mateus Da Silva Marans**

**Cuiabá**

**2024**

**Mateus Da Silva Marans**

**ALGORITMOS E LÓGICA DE PROGRAMAÇÃO**

**Trabalho apresentado no Sesc escola, curso de jogos digitais, Senac.**

**Orientador: Professor Wanderson**

**Cuiabá**

**2024**

**SUMÁRIO**

1. **Algoritmo.**
   1. Definição de algoritmos e sua importância na programação.
   2. Lógica de programação e sua relação com algoritmos.
2. **Estruturas de controle.**
   1. Pesquisa sobre estruturas de controle como sequência, seleção e repetição.
   2. Exemplos de algoritmos que utilizam estruturas de controle.
3. **Tipos de dados e variáveis**
   1. Definição de tipos de dados básicos, como inteiros, ponto flutuante, booleanos, etc.
   2. Exemplos de declaração e utilização de variáveis em algoritmos.
4. **Funções e modularização**
   1. Pesquisa sobre o conceito de função na programação.
   2. A importância da modularização na escrita de algoritmos.
   3. Exemplos de funções simples e sua utilização em algoritmos mais complexos.
5. **Algoritmo**

Um algoritmo é uma sequência finita de etapas bem definidas e inequívocas que levam a uma solução para um problema específico. Algoritmos são usados ​​em diversas áreas, como ciência da computação, matemática, engenharia e outras disciplinas. Eles são a base para a criação de programas de computador e são essenciais para automatizar tarefas e resolver problemas complexos.

* 1. **Definição de algoritmos e sua importância na programação**

Eficiência: Um algoritmo que resolve problemas de forma eficiente e utiliza recursos mínimos, como tempo de processamento e memória. Ao escolher entre diferentes métodos para resolver um problema, a eficiência do algoritmo é uma consideração importante. Reutilização: Os algoritmos desenvolvidos podem ser utilizados em diversas situações e finalidades. Isso economiza tempo e esforço, pois não é necessário reescrever a solução sempre que surge um problema semelhante. Implementação de software: Depois que um algoritmo é projetado e testado, ele pode ser implementado como código de projeto. Esta implementação pode ser feita em diferentes linguagens dependendo das necessidades e exigências do projeto. Simplificando, os algoritmos são o núcleo do assunto e fornecem métodos e métodos sistemáticos para resolver problemas computacionais.

* 1. **Lógica de programação e sua relação com algoritmos**

Lógica é um conjunto de regras e métodos usados ​​para desenvolver algoritmos eficientes e corretos. Identificar soluções: Os programadores podem usar os princípios da lógica de programação para identificar a solução mais eficiente e elegante para um problema. Em outras palavras, a lógica de programação fornece os princípios e técnicas necessários para desenvolver algoritmos eficientes e otimizados. Isso ajuda os programadores a pensar de forma lógica e estrutural ao lidar com problemas de computador e a garantir que suas soluções sejam claras, corretas e eficazes.

1. **Estruturas de controle**

Estruturas de controle são elementos fundamentais da lógica de programação e são usadas para controlar o fluxo de programação. Estrutura sequencial: A estrutura sequencial é a forma mais simples de controle de fluxo e consiste em uma sequência linear de instruções que são executadas individualmente na ordem em que aparecem no código. Estruturas (ou métodos) de decisão: As estruturas de decisão permitem que um programa siga diferentes cursos de ação com base em condições ou expressões booleanas. Essas estruturas organizacionais podem ser combinadas e aninhadas de diversas maneiras para criar mudanças complexas que atendam a diferentes requisitos. Por exemplo, é comum usar estruturas de decisão em uma estrutura de repetição para controlar o comportamento do programa com base em diferentes condições durante a execução. Isso permite que os programadores controlem o fluxo da programação e tomem decisões com base em diferentes situações, resolvendo assim diversos problemas de software.

* 1. **Pesquisa sobre estruturas de controle como sequência, seleção e repetição**

Os controles, também chamados de estruturas de processo, são importantes na programação de computadores. Ao usar comandos em um programa, os desenvolvedores podem controlar o fluxo de comandos de determinadas maneiras. As instruções são executadas individualmente e aparecem no código, sem exceções ou bifurcações no fluxo de trabalho. A estrutura é representada por uma lista sequencial de instruções, na qual cada linha de código é executada. Decisão (ou escolha): O processo de tomada de decisão permite que um programa execute diferentes ações com base em informações especificadas. Isso é feito usando cláusulas condicionais como if, if, etc. Calcular a idade do usuário Mostrar "Você é legal" Toque "Você ainda é jovem" Loops (ou Loops): Os loops são projetados para permitir que o código seja executado quando certas condições forem atendidas. Esses recursos podem ser combinados e implementados para criar estilos que atendam a diferentes necessidades. Os programadores podem controlar o fluxo do programa e tomar decisões com base em diferentes cenários para resolver diferentes problemas do computador.

* 1. **Exemplos de algoritmos que utilizam estruturas de controle**

Aqui estão alguns exemplos simples de algoritmos usando distribuição, distribuição e iteração: A média de três anos é calculada como: Este é um exemplo de algoritmo sequencial, que executa instruções na ordem em que aparecem. Como verificar se é antigo ou novo: 4. Exiba "Este número é o mesmo" 6. Exibir "Quantidade é diferente" Neste algoritmo, um método de seleção (se-então) é usado para selecionar a mensagem a ser exibida com base no número de bits. Este algoritmo usa métodos de conversão (em loop) para representar números naturais de 1 a 10. Como calcular a soma dos números entre 1 e 100: 2. O número do índice começa em 0 3. Comece a mudar (i) para 1 3. Comece a mudar (i) para 1 7.Finalmente 7. fim do tempo O algoritmo também usa estruturas temporárias para somar números entre 1 e 100. Estes são exemplos básicos que mostram como usar controle, seleção e iteração em algoritmos simples.

1. **Tipos de dados e variáveis**

Tipos de dados e variáveis ​​são componentes fundamentais de qualquer linguagem de programação. Um tipo de dados define o tipo de valor que uma variável pode armazenar. String (str): Armazena strings como "Hello World!", "Python" e "123". Valor booleano (bool): Armazena o valor lógico verdadeiro (True) ou falso (False). Lista: armazena uma coleção ordenada de itens que podem ter diferentes tipos de dados. As variáveis ​​possuem tipos de dados associados que determinam os tipos de valores que podem armazenar. imprimir(float) # Saída: 3.14 print(caractere) # Saída: a print(texto) # Saída: Olá, mundo! print(boolean) # Saída: Verdadeiro Este exemplo Python declara variáveis ​​de diferentes tipos e atribui valores a elas. Estes são apenas alguns dos tipos de dados e variáveis ​​mais comuns.

**3.1 Definição de tipos de dados básicos, como inteiros, ponto flutuante, booleanos, etc**

Inteiros (int):

Os inteiros são números inteiros, ou seja, números sem parte fracionária.

Eles podem ser positivos, negativos ou zero.

Exemplos de inteiros são: -10, 0, 42, 100, etc.

Em muitas linguagens de programação, os inteiros têm limites em relação ao valor máximo e mínimo que podem ser representados.

Ponto Flutuante (float):

Os números de ponto flutuante representam números reais, incluindo números com parte fracionária.

Eles podem ser positivos, negativos ou zero.

Exemplos de números de ponto flutuante são: -3.14, 0.5, 1.618, etc.

Os números de ponto flutuante são representados usando notação científica (por exemplo, 6.02e23) ou ponto decimal (por exemplo, 3.14159).

Booleanos (bool):

Os booleanos representam valores lógicos verdadeiro (True) ou falso (False).

São comumente usados para expressar condições em operações de controle de fluxo, como declarações condicionais e loops.

Exemplos de expressões booleanas são: True, False, 5 > 3 (resulta em True), 10 == 5 (resulta em False), etc.

Esses são os tipos de dados básicos mais comuns encontrados na maioria das linguagens de programação. Eles fornecem os blocos de construção essenciais para a manipulação de dados e a tomada de decisões em programas de computador.

**3.2 Exemplos de declaração e utilização de variáveis em algoritmos**

Exemplo de declaração e utilização de uma variável inteira em Python: python Copy code # Declaração e utilização de uma variável inteira numero = 10 dobro = numero \* 2 print("O dobro de", numero, "é", dobro) Exemplo de declaração e utilização de uma variável de ponto flutuante em C: c Copy code #include <stdio.h> int main() { // Declaração e utilização de uma variável de ponto flutuante float altura = 1.75; printf("A altura é %.2f metros.\n", altura); return 0; } Exemplo de declaração e utilização de uma variável booleana em Java: java Copy code public class Main { public static void main(String[] args) { // Declaração e utilização de uma variável booleana boolean isSunny = true; if (isSunny) { System.out.println("O tempo está ensolarado."); } else { System.out.println("O tempo está nublado."); } } } Exemplo de declaração e utilização de uma variável string em JavaScript: javascript Copy code // Declaração e utilização de uma variável string let nome = "João"; console.log("Olá, " + nome + "!");

**4 Funções e modularização**

Uma função é um bloco de código que executa uma tarefa específica quando chamada. As funções são úteis para dividir o código em partes menores e mais gerenciáveis, tornando o código mais fácil de manter e reutilizar. Um exemplo de definição de função Python: Neste exemplo, sum é o nome da função e leva dois parâmetros a e b. Em linguagens de programação como Python, a modularização geralmente é obtida através da criação de arquivos de módulos separados, onde cada arquivo contém um conjunto de funções relacionadas. Um exemplo de modularização em Python: imprimir (sumário\_resultado) # Resultado: 8 print (resultado\_subtracao) # Resultado: 6 Neste exemplo, as funções de adição e subtração estão definidas no módulo operacoes.py.

**4.1 Pesquisa sobre o conceito de função na programação.**

Extração e reutilização: As funções permitem escrever o mesmo código e reutilizá-lo muitas vezes em diferentes partes do programa. Isto incentiva a reutilização de código e evita duplicação, tornando a codificação mais eficiente e fácil de manter. Cada função é responsável por uma tarefa específica, tornando o código mais fácil de entender e mais flexível. Parâmetros e argumentos: as funções podem aceitar parâmetros, que são informações fornecidas quando a função é chamada. Valor de retorno: uma função pode retornar um valor calculado ou processado para o código de chamada. Isso permite que as tarefas transmitam resultados e dados estatísticos ao longo do projeto.

**4.2 A importância da modularização na escrita de algoritmos**.

Organização do código: Algoritmos complexos podem ser divididos em módulos menores e mais gerenciáveis. Cada módulo é responsável por uma parte específica do algoritmo, tornando o código mais fácil de entender, manter e depurar. Isso permite que os desenvolvedores se concentrem no uso de cada módulo sem precisar entender os detalhes internos da implementação. Em resumo, a modularidade é uma prática essencial na criação de algoritmos porque facilita a organização, reutilização, manutenção e colaboração eficazes no desenvolvimento de software.

**4.3 Exemplos de funções simples e sua utilização em algoritmos mais complexos.**

Exemplo 1: Função para calcular a média de uma lista de números Neste exemplo, a função calculator\_media pega uma lista de números como entrada e retorna a média desses números. Esta função pode ser facilmente utilizada em algoritmos mais complexos que requerem cálculos utilizando valores médios. Exemplo 2: Função para verificar se um número é par print(num, “é um número par.”) print(num, “Não é um número par.”) Exemplo 3: Função que calcula o fatorial de um número n\* retorna a classificação calculada (n-1). Resultado = classificação de cálculo (número) Neste exemplo, a função calcula\_fatorial recebe um número inteiro n como entrada e retorna o fatorial desse número.

**Referências:**

chat.openai.com