

**SESC ESCOLA CUIABÁ**

**SAMUEL JOÃO DA COSTA**

ALGORITMO E LÓGICA DE PROGAMAÇÃO

**CUIABÁ**

2024

****

**SAMUEL JOÃO DA COSTA**

**ALGORITMO E LÓGICA**

DE PROGAMAÇÃO

trabalho apresentado pelo curso de

jogos digitais

orientador: wanderson

**CUIABÁ**

2024

**SUMÁRIO**

1 ALGORITMO....................................................................................................................................................4

* 1. DEFINIÇÃO DE ALGORITMOS E SUA IMPORTÂNCIA NA PROGAMAÇÃO
  2. LÓGICA DE PROGAMAÇÃO E SUA RELAÇÃO COM ALGORITMOS

1. ESTRUTURAS DE CONTROLE...................................................................................................................................................5
   1. PESQUISA SOBRE ESTRUTURAS DE CONTROLE COMO SEQUÊNCIA, SELEÇÃO E REPETIÇÃO
   2. EXEMPLOS DE ALGORITMOS QUE UTILIZAM ESTRUTURAS DE CONTROLE
2. TIPOS DE DADOS E VARIÁVEIS..................................................................................................................................................7
   1. DEFINIÇÃO DE TIPOS DE DADOS BÁSICOS, COMO INTEIROS, PONTO FLUTUANTE, BOOLEANOS, ETC.
   2. EXEMPLOS DE DECLARAÇÃO E UTILIZAÇÃO DE VARIAVÉIS EM ALGORITMOS
3. FUNÇÕES E MODULARIZAÇÃO...................................................................................................................8
   1. PESQUISA SOBRE O CONCEITO DE FUNÇÃO NA PROGAMAÇÃO
   2. A IMPORTÂNCIA DA MODULARIZAÇÃO NA ESCRITA DE ALGORITMOS
   3. EXEMPLOS DE FUNÇÕES SIMPLES E SUA UTILIZAÇÃO EM ALGORITMOS MAIS COMPLEXOS

**RESUMO**

Um algoritmo na programação é uma sequência de passos lógicos e bem definidos para resolver um problema. Ele define como os dados são processados para produzir um resultado desejado. Um exemplo simples é encontrar o maior número em uma lista: defina uma lista, escolha o primeiro número como o maior, depois compare cada número na lista com o maior até encontrar o maior de todos.

**DEFINIÇÕES E SUA IMPORTÂNCIA**

Algoritmos são conjuntos de instruções bem definidas e organizadas que descrevem um processo ou solução para um problema. Eles são fundamentais na programação porque servem como a base para o desenvolvimento de software. Algoritmos ajudam os programadores a:

Resolver Problemas: Algoritmos permitem aos programadores transformar problemas complexos em uma série de passos mais simples e compreensíveis.

Planejar Lógica: Eles ajudam a planejar a lógica de um programa antes de escrever o código. Isso ajuda a entender o problema em um nível abstrato antes de começar a implementação.

Eficiência: Algoritmos bem projetados podem melhorar a eficiência do software, garantindo que ele execute tarefas de forma rápida e com o mínimo de recursos.

Reutilização de Código: Algoritmos bem definidos podem ser reutilizados em diferentes partes do código ou mesmo em diferentes projetos, economizando tempo e esforço de desenvolvimento.

Manutenção: Algoritmos claros e bem documentados facilitam a manutenção do código ao longo do tempo, pois outros programadores podem entender facilmente o que está sendo feito e fazer alterações conforme necessário.

**RELAÇÃO ENTRE LÓGICAS DE PROGAMAÇÕES E OS ALGARITMOS**

Pensamento Lógico: A lógica de programação envolve a habilidade de pensar de forma lógica e estruturada para resolver problemas. Os algoritmos são a manifestação dessa lógica em uma sequência de passos que podem ser seguidos para alcançar um objetivo.

Estruturação de Problemas: A lógica de programação ajuda a estruturar um problema de forma clara e compreensível, identificando as etapas necessárias para resolvê-lo. Os algoritmos são então criados com base nessa estruturação, descrevendo passo a passo como resolver o problema.

Implementação Prática: Uma vez que a lógica de programação é usada para entender e estruturar um problema, os algoritmos são desenvolvidos como uma implementação prática dessa lógica em uma linguagem de programação específica.

Dependência Mútua: Embora a lógica de programação seja uma habilidade fundamental para a criação de algoritmos eficazes, a prática na criação de algoritmos também ajuda a desenvolver e aprimorar a lógica de programação.

**RESUMO**

As estruturas de controle na programação são ferramentas essenciais para controlar o fluxo de execução do código. Elas incluem estruturas condicionais (como IF, ELSE e SWITCH-CASE) para tomada de decisões, estruturas de repetição (como FOR, WHILE e DO-WHILE) para execução de blocos de código repetidamente, e estruturas de controle de fluxo (como BREAK, CONTINUE e RETURN) para modificar o comportamento do programa. Essas estruturas são fundamentais para criar algoritmos eficientes e funcionais em diversas linguagens de programação.

**ESTRUTURAS DE CONTROLE**

Sequência: Na estrutura de sequência, as instruções são executadas uma após a outra, em ordem linear, sem qualquer desvio no fluxo de execução. Isso significa que cada instrução é executada após a anterior, sem considerar condições ou repetições.

Seleção (ou Condicional): A estrutura de seleção permite que diferentes blocos de código sejam executados com base em condições específicas. Por exemplo, o código pode verificar se uma condição é verdadeira e, se for, executar um bloco de código específico. Caso contrário, pode executar um bloco alternativo.

Repetição (ou Iteração): A estrutura de repetição permite que um bloco de código seja executado repetidamente enquanto uma condição específica for verdadeira. Isso é útil quando você precisa executar a mesma tarefa várias vezes sem ter que escrever o código repetidamente.

Essas estruturas de controle são fundamentais na programação, pois permitem que os programadores controlem o fluxo de execução do programa de acordo com as condições e requisitos específicos do problema que estão tentando resolver.

**ALGORITMOS QUE UTILIZAM ESTRUTURAS DE CONTROLE**

**Algoritmo para determinar se um número é par ou ímpar:**

mathematica

Início

Ler o número

Se o número % 2 == 0

Escrever "O número é par"

Senão

Escrever "O número é ímpar"

Fim

**Algoritmo para encontrar o maior número em uma lista:**

markdown

Início

Definir uma lista de números

Definir um valor máximo como o primeiro número da lista

Para cada número na lista

Se o número atual for maior que o valor máximo

Atualizar o valor máximo para ser o número atual

Escrever o valor máximo

Fim

**Algoritmo para calcular a média de uma lista de números:**

css

Início

Definir uma lista de números

Definir uma variável soma como zero

Para cada número na lista

Adicionar o número à soma

Calcular a média dividindo a soma pelo total de números na lista

Escrever a média

Fim

Esses são apenas alguns exemplos simples de algoritmos que utilizam estruturas de controle como sequência, seleção (condicional) e repetição (iteração) para realizar tarefas específicas. Essas estruturas são fundamentais para criar algoritmos mais complexos e resolver uma variedade de problemas na programação.

**RESUMO**

Tipos de dados e variáveis são fundamentais na programação para armazenar e manipular informações. Os tipos de dados incluem primitivos (como inteiros, ponto flutuante, caracteres e booleanos) e compostos (como strings, arrays, estruturas, listas e dicionários). Variáveis são nomes que se referem a valores armazenados na memória do computador e devem ser declaradas com um tipo de dado específico antes de serem utilizadas.

**DEFINIÇÃO DE TIPOS DE DADOS E EXEMPLOS**

Tipos de Dados Primitivos:

Inteiro (int): Representa números inteiros, como -5, 0, 10.

Ponto Flutuante (float ou double): Representa números reais (com ponto decimal), como 3.14, -0.5, 2.0.

Caractere (char): Representa um único caractere, como 'A', 'b', '$'.

Booleano (bool): Representa valores verdadeiro (true) ou falso (false).

Tipos de Dados Compostos:

String: Sequência de caracteres, como "Olá, mundo!", "1234".

Array: Coleção ordenada de elementos do mesmo tipo, como [1, 2, 3, 4], ["maçã", "banana", "laranja"].

Estrutura (struct): Agrupa diferentes tipos de dados em uma única unidade, como um registro.

Lista (list): Coleção dinâmica de elementos de qualquer tipo.

Dicionário (dictionary): Coleção de pares chave-valor, onde cada valor é acessado através de uma chave.

Variáveis:

Uma variável é um nome que se refere a um valor armazenado na memória do computador.

Antes de usar uma variável, é necessário declará-la, indicando o tipo de dado que ela vai armazenar.

Exemplos de declaração e uso de variáveis:

python

inteiro = 10

flutuante = 3.14

caractere = 'A'

booleano = True

string = "Olá, mundo!"

Esses são os principais tipos de dados e variáveis em programação, essenciais para armazenar e manipular informações durante a execução de um programa.

**RESUMO**

Funções são blocos de código que executam tarefas específicas e promovem a reutilização de código. Elas são definidas com um nome, podem receber parâmetros e podem retornar valores. Modularização é a prática de dividir um programa em partes menores e independentes, cada uma responsável por uma função específica. Isso torna o código mais organizado, legível e fácil de manter, além de facilitar a reutilização e colaboração entre membros da equipe.

**FUNÇÕES E MODULARIZAÇÃO**

Funções:

Uma função é um bloco de código que realiza uma tarefa específica.

Elas são definidas com um nome, uma lista de parâmetros (opcionais) e um corpo de código.

As funções podem retornar um valor, que pode ser usado em outras partes do programa.

Elas promovem a reutilização de código, tornando-o mais legível e fácil de manter.

Exemplo de uma função em Python que retorna a soma de dois números:

Python

def soma(a, b):

return a + b

Modularização:

Modularização é o processo de dividir um programa em partes menores e independentes, chamadas de módulos ou funções.

Cada módulo ou função realiza uma tarefa específica, tornando o programa mais organizado e fácil de entender.

Isolar partes do código em funções permite que elas sejam desenvolvidas e testadas separadamente.

Isso também facilita a colaboração entre os membros da equipe, pois diferentes pessoas podem trabalhar em diferentes partes do código ao mesmo tempo.

Exemplo de modularização em Python, onde uma função é definida para calcular a área de um círculo:

python

import math

def area\_do\_circulo(raio):

return math.pi \* raio\*\*2

Em resumo, funções são blocos de código que executam uma tarefa específica e podem ser reutilizadas em diferentes partes do programa, promovendo a modularização, que é a prática de dividir o código em partes menores e independentes para torná-lo mais organizado e fácil de entender.

**Fonte:**

ChatGPT, OPENAI