Miniguia para programar em C

1 Fundamentos Iniciais

O que é a programação? A programação é a arte de criar instruções literais e detalhadas para que um computador execute tarefas específicas. Esses comandos são escritos em linguagens de programação, que seguem regras e símbolos precisos.

Por que a linguagem C? A linguagem C tem estrutura simples e direta permite a compreensão de conceitos fundamentais da programação de forma clara e objetiva. Assim, habilidades essenciais como o controle direto sobre o hardware do computador e o gerenciamento manual da memória, que são o que proporciona um entendimento mais profundo sobre como os programas funcionam internamente, são aprendidas (mesmo que forçadamente).

A eficiência e a ampla aplicação da linguagem C em diversas áreas, como sistemas operacionais e sistemas embarcados, a tornam uma escolha sólida para uma primeira linguagem, construindo assim uma base sólida para seus estudos.

2 Algoritmos

2.1 Definição de algoritmos

Um algoritmo é uma sequência finita e ordenada de passos que, quando executados, levam à resolução de um problema específico. No mundo da computação, os algoritmos são a base para a criação de programas e softwares porque guiam o computador através de uma série de etapas para realizar tarefas complexas, desde cálculos matemáticos até a busca de informações em grandes bancos de dados.
A importância dos algoritmos reside na sua capacidade de organizar o pensamento e otimizar a resolução de problemas. Ao descrever um problema em forma de algoritmo, torna-se mais fácil a sua compreensão, assim como comunicá-lo a outras pessoas e implementá-lo em um programa de computador. Um bom algoritmo é preciso, eficiente, finito e capaz de lidar com diferentes tipos de informações.
A criação de algoritmos envolve a habilidade de decompor um problema complexo em partes menores e mais simples, cada parte podendo então ser resolvida por um conjunto de instruções específicas. No dia a dia, algoritmos são usados constantemente, mesmo que inconscientemente. Por exemplo, ao fazer um sanduíche:

```
Início
1. Separar duas fatias de pão
2. Pegar a maionese da geladeira
3. Abrir o pote de maionese e passá-la no pão
4. Adicionar duas fatias de mortadela
5. Adicionar cinco folhas de rúcula
6. Juntar os pães
Fim
```

3 Lógica de Programação

3.1 Pensamento lógico

3.1.1 Decomposição de problemas

A decomposição de problemas é uma técnica fundamental na lógica de programação, e consiste em dividir um problema complexo em subproblemas menores e mais fáceis de resolver.
Após decompor um problema, é crucial organizar os passos em uma sequência lógica, considerando a ordem de dependência entre as tarefas. Por exemplo, num programa que calcula a média de três notas, essas notas devem ser informadas pelo usuário antes de serem utilizadas nos cálculos em si. Além da ordem, a eficiência também é importante: qual a melhor maneira de realizar cada tarefa e quais recursos são necessários para cada etapa?

```
1. Obter primeira nota
2. Obter segunda nota
3. Obter terceira nota
4. Somar as três notas
5. Dividir a soma por 3
6. Mostrar o resultado
```

3.1.2 Sequência lógica de passos

A sequência lógica dos passos em um algoritmo é crucial para garantir que o problema seja resolvido de forma correta. Ao definir a ordem das tarefas, é preciso considerar a dependência entre elas, ou seja, quais tarefas precisam ser concluídas antes de iniciar outras. Além disso, a eficiência também deve ser levada em conta, buscando a melhor forma de realizar cada etapa. Ao organizar os passos de forma lógica, evitamos erros e otimizamos o processo de resolução do problema.

Recapitulando

A decomposição de problemas e a organização sequencial dos passos são habilidades essenciais para quem deseja se tornar um bom programador. Ao dominar essas técnicas, você será capaz de enfrentar desafios complexos de forma mais organizada e eficiente.

4 Pseudocódigo

Antes de começar a programar em C, o passo sequencial lógico é de utilizar alguma ferramenta que busca organizar e expressar um dado algoritmo de forma clara e concisa. Assim, cria-se um rascunho que geralmente utiliza uma linguagem híbrida que combina elementos da língua comum com estruturas típicas de programação.
Ao escrever em pseudocódigo, utilizam-se palavras-chave como `INÍCIO`, `FIM`, `SE`, `ENTÃO`, `SENÃO` e `ENQUANTO` para estruturar o código. Cada linha do pseudocódigo deve conter uma única instrução para facilitar a leitura e a compreensão. A indentação é crucial para visualizar a hierarquia entre as instruções, mostrando quais comandos estão subordinados a outros.

```
INÍCIO
    ESCREVA "Digite um número: "
    LEIA numero

    SE (numero MOD 2 = 0) ENTÃO
        ESCREVA "O número é par"
    SENÃO
        ESCREVA "O número é ímpar"
    FIM_SE
FIM
```

A grande vantagem do pseudocódigo é sua flexibilidade, que permite que o programador se concentre na lógica do algoritmo sem se preocupar com os detalhes sintáticos de uma linguagem de programação específica, como o C. Dessa forma, é possível imaginar um algoritmo mais facilmente.
Não apenas isso, mas o pseudocódigo serve como uma excelente ferramenta de comunicação, pois permite que programadores e não-programadores discutam e entendam a solução de um problema de forma colaborativa.

Recapitulando

O pseudocódigo é uma linguagem intermediária que facilita a transição entre a ideia e a implementação de um algoritmo, utilizando-se de elementos da língua comum com estruturas típicas da programação.

5 Antes do C...

5.1 Bibliotecas

Antes de começar a programar em C, saiba que seus programas deverão iniciar da seguinte forma:

```
#include <stdio.h>

int main()
{
    // Código a ser executado

    return 0;
}
```

6 Tipos de dados

6.1 Variáveis

As variáveis possuem um nome e um tipo de dado específico, indicando o tipo de informação que ela pode armazenar (como números, letras ou valores lógicos). Na prática, elas são como "caixas" utilizadas para armazenar dados num dado programa.
O termo empregado na programação é o de *declarar* uma variável, e esse processo requer definir seu tipo de dado. Uma variável pode ser declarada e ter a si um valor inicial naquele momento ou posteriormente. Confira a sintaxe básica da declaração de variáveis em C:

```
tipo nome = valor;  // Com inicialização
tipo nome;          // Sem inicialização
```

O nome de uma variável é arbitrário e é considerada uma boa prática que seja descritivo e reflita o propósito da variável. É importante seguir algumas convenções de nomenclatura, como começar o nome com uma letra ou underscore e evitar o uso de palavras reservadas da linguagem. Por exemplo, separar as palavras por underscores ou utilizar a primeira letra das segundas palavras em maiúsculo:

```
int variavelPrimeira = 25;
float variavelSegunda;
```

Recapitulando

Variáveis são elementos fundamentais da programação que permitem a manipulação de dados de forma organizada e eficiente.

6.1.1 Números inteiros

Os números inteiros do tipo `int` são utilizados para armazenar valores numéricos inteiros, ou seja, números sem parte decimal.

```
int idade = 25;
```

6.1.2 Números decimais

Os números decimais podem ser representados pelos tipos `float` e `double`. `float` oferece precisão simples, enquanto `double` oferece precisão dupla e por isso é mais preciso e deve ser utilizado com cálculos mais precisos.

```
float preco = 10.99f;
double pi = 3.14159265359;
```

6.1.3 Caracteres

Os caracteres são utilizados para armazenar um único caractere, como uma letra ou dígito. **Um caractere é delimitado por aspas simples.**

```
char letra = 'A';
```

6.1.4 Booleanos

Como a linguagem C não possui um tipo booleano nativo, é utilizado o tipo `int` representar valores lógicos. O valor 0 representa falso e qualquer outro valor diferente de zero representa verdadeiro.

```
int verdadeiro = 1;
int falso = 0;
```

É interessante notar que a escolha do tipo de dado também influencia a eficiência do programa e utilizar um tipo de dado maior do que o necessário pode consumir mais memória e tornar o programa mais lento.

Recapitulando

Cada tipo de dado possui um tipo de informação que se deseja armazenar e um tamanho específico em memória e um intervalo de valores limitado.

6.2 Operadores

6.2.1 Operadores aritméticos

Os operadores aritméticos são utilizados para realizar operações matemáticas básicas, como adição, subtração, multiplicação, divisão e cálculo do resto da divisão (módulo):

```
+ // Adição
- // Subtração
* // Multiplicação
/ // Divisão
% // Resto da divisão (módulo)
```

6.2.2 Operadores relacionais

Os operadores relacionais são utilizados para comparar valores e obter resultados booleanos (verdadeiro ou falso) e são essenciais para criar expressões condicionais:

```
==  // Igual a
!=  // Diferente de
>   // Maior que
<   // Menor que
>=  // Maior ou igual
<=  // Menor ou igual
```

6.2.3 Operadores lógicos

Os operadores lógicos são utilizados para combinar expressões booleanas e criar expressões lógicas mais complexas:

```
&&  // E (AND)
||  // OU (OR)
!   // NÃO (NOT)
```

Não se esqueça que a ordem dos operadores influencia os resultados das expressões.

Recapitulando

Operadores são ferramentas poderosas que permitem manipular dados e tomar decisões em seus programas e ao conhecer bem os diferentes tipos de operadores e suas funcionalidades, será possível escrever códigos mais eficientes.

7 Estruturas de controle

Em C, as estruturas de controle são essenciais para criar programas capazes de tomar decisões com base em diferentes condições.

7.1 Estrutura Sequencial

A estrutura sequencial é a mais simples e básica, pois as instruções são executadas uma após a outra, na ordem em que aparecem no código.

7.1.1 Entrada de dados

A função `scanf` permite que você leia dados do teclado e armazene-os em variáveis.

```
scanf("%tipo", &variavel);

int idade;
float altura;
char inicial;

scanf("%d", &idade);    // Lê um inteiro
scanf("%f", &altura);   // Lê um float
scanf("%c", &inicial);  // Lê um caractere

// Lê múltiplos valores
scanf("%d %f", &idade, &altura);
```

7.1.2 Saída de dados

A função `printf` é utilizada para exibir dados na tela e requer especificadores de formato que indicam o tipo de dado que será exibido. Por exemplo, `%d` para inteiros, `%f` para números de ponto flutuante, `%c` para caracteres e `%s` para strings.

```
printf("texto %tipo", variavel);

printf("%d", numero);    // Inteiro

printf("%f", decimal);   // Float/Double

printf("%.2f", preco);  // Float com 2 casas decimais

printf("%c", letra);    // Caractere

printf("%s", texto);    // String

// Exemplo combinado
int idade = 25;
float altura = 1.75;
printf("Idade: %d anos\nAltura: %.2f m", idade, altura);
```

7.2 Estruturas de seleção

As estruturas de seleção ou "condicionais" são mecanismos que permitem tomar decisões com base em condições específicas.

7.2.1 If/else

A estrutura `if/else` é a mais versátil e flexível dentre os condicionais, porque permite executar um bloco de código caso uma determinada condição seja verdadeira e, opcionalmente, outro bloco de código caso a condição seja falsa. A sintaxe básica é:

```
if (condição) {
    // Código a ser executado se a condição for verdadeira
} else {
```

```
    // Código a ser executado se a condição for falsa
}
```

Isso pode ser extendido, como neste exemplo:

```
if (nota >= 9) {
    printf("A");
} else if (nota >= 7) {
    printf("B");
} else if (nota >= 5) {
    printf("C");
} else {
    printf("D");
}
```

7.2.2 Switch/case

A estrutura `switch/case` é mais adequada para situações quando se precisa comparar uma variável com um conjunto de valores constantes e é geralmente mais eficiente que `if/else` quando há muitas condições a serem verificadas. Sua sintaxe básica é:

```
switch (expressão) {
    case valor1:
        // Código a ser executado se a expressão for igual a valor1
        break;
    case valor2:
        // Código a ser executado se a expressão for igual a valor2
        break;
    default:
        // Código a ser executado se nenhum dos casos corresponder
}
```

A palavra-chave `break` é essencial para evitar que o programa continue executando os casos seguintes após encontrar uma correspondência. O `default` é opcional e é executado caso nenhum dos casos seja verdadeiro.

Recapitulando
As estruturas de seleção são ferramentas essenciais para criar programas que tomam decisões com base em diferentes condições e a escolha entre `if/else` e `switch/case` depende da natureza do problema a ser resolvido e da complexidade das condições a serem verificadas.

Sugestão de prática

- Escreva um algoritmo em C que leia três valores inteiros diferentes e mostre-os em ordem decrescente.
- Elabore um algoritmo que receba como entrada um ano e retorne a ele a informação se ele é um ano bissexto ou não.
- Escreva um algoritmo em C que leia um número inteiro positivo de três dígitos e verifique se a soma dos dígitos desse número é igual ao produto dos dígitos.
- Escreva um programa em C que pede todos os lados de um triângulo e determina se ele é equilátero, isósceles ou escaleno.

7.3 Estruturas de repetição

As estruturas de repetição (também conhecidas como laços) são mecanismos que permitem executar um bloco de código repetidamente enquanto uma determinada condição for verdadeira.

7.3.1 While

Trata-se do laço mais básico e versátil. Ele executa um bloco de código enquanto uma condição especificada for verdadeira e e essa condição é verificada no início de cada iteração. Se for verdadeira, o bloco de código é executado e a condição é verificada novamente. Caso contrário, o laço é encerrado.

```
while (condicao) {
    // código a ser repetido
}

// Exemplo: Contagem regressiva
int contador = 5;
while (contador > 0) {
    printf("%d ", contador);
    contador--;
}
```

7.3.2 Do/while

Semelhante ao `while`, mas com uma diferença crucial: a condição é verificada ao final de cada iteração. Isso garante que o bloco de código seja executado pelo menos uma vez, mesmo que a condição inicial seja falsa.

```
do {
    // código a ser repetido
} while (condicao);

// Exemplo: Menu
do {
    printf("\n1-Continuar 0-Sair\n");
    scanf("%d", &opcao);
} while (opcao != 0);
```

7.3.3 For

Este é o laço ideal para situações em que o número de iterações é conhecido e possui três partes: inicialização, condição e incremento. A inicialização é executada uma vez antes do início do laço, a condição é verificada no início de cada iteração e o incremento é executado ao final de cada iteração.

```
// Sintaxe básica
for (inicializacao; condicao; incremento) {
    // código a ser repetido
}

// Exemplo: Tabuada
for (int i = 1; i <= 10; i++) {
    printf("%d x 5 = %d\n", i, i * 5);
}
```

Recapitulando
Estruturas de repetição são ferramentas poderosas que permitem automatizar tarefas repetitivas e criar programas mais eficientes.

Sugestão de prática

- Escreva um programa que imprima a tabuada de um número inteiro positivo.
- Escreva um programa que calcule a soma dos dígitos de um número inteiro positivo.
- O fatorial de um número natural n é o produto de todos os números naturais de 1 até n. Escreva um programa em C que calcule o fatorial de um número inteiro positivo.
- Escreva um programa que inverta a ordem dos dígitos de um número inteiro positivo.
- Escreva um programa em C que, dado um número inteiro positivo N, determine se existe algum número perfeito entre 1 e N. Lembre-se que um número perfeito é um número inteiro positivo igual à soma de seus divisores próprios positivos. Por exemplo, 6 é um número perfeito porque $1 + 2 + 3 = 6$.
- Escreva um programa em C que leia um número inteiro positivo e verifique se ele é um número de Armstrong. Um número de Armstrong é um número inteiro de n dígitos que é igual à soma das n-ésimas potências de seus dígitos. Por exemplo, 153 é um número de Armstrong porque $1\hat{}3 + 5\hat{}3 + 3\hat{}3 = 153$.
- Elabore um algoritmo em C que mostre todos os fatores primos de um número inteiro positivo N dado como entrada pelo usuário. A saída do seu programa deve ser um sequência de linhas, com cada linha contendo um fator primo de N, e esses fatores podem se repetir mas devem ser mostrados em ordem não-decrescente.

8 Vetores e matrizes

8.1 Vetores

Em C, um vetor é uma estrutura que armazena elementos do mesmo tipo em posições consecutivas de memória e são acessados através de um índice que começa em zero. A declaração pode ser feita especificando o tamanho entre colchetes, e a inicialização pode ocorrer na própria declaração ou posteriormente. Por exemplo:

```
int numeros[5];                      // Declaração simples
int valores[5] = {1, 2, 3, 4, 5};  // Declaração com inicialização
int automatico[] = {1, 2, 3};      // Tamanho definido automaticamente
```

O acesso aos elementos é feito usando o operador de índice [], e as operações mais comuns incluem inserção, remoção e busca. Considere este exemplo de manipulação de vetores:

```
int notas[5] = {85, 92, 78, 95, 88};
int soma = 0;

// Percorrendo e somando elementos
for(int i = 0; i < 5; i++) {
    soma += notas[i];
}

float media = soma / 5.0;
```

```
// Buscando maior nota
int maior = notas[0];
for(int i = 1; i < 5; i++) {
    if(notas[i] > maior) {
        maior = notas[i];
    }
}
```

8.2 Matrizes

Uma matriz é um vetor bidimensional, organizado em linhas e colunas e por isso a sua declaração requer a especificação de ambas as dimensões, e a memória é alocada de forma adjascente, sendo acessada através de dois índices. Sua sintaxe básica é:

```
int matriz[3][3];                                    // Declaração
int matriz[2][3] = {{1, 2, 3}, {4, 5, 6}};          // Com inicialização
int matriz[][3] = {{1, 2, 3}, {4, 5, 6}};           // Linhas automáticas
```

As operações com matrizes frequentemente envolvem dois loops aninhados. Por exemplo:

```
// Soma de duas matrizes 3x3
int m1[3][3] = {{1,2,3}, {4,5,6}, {7,8,9}};
int m2[3][3] = {{9,8,7}, {6,5,4}, {3,2,1}};
int resultado[3][3];

for(int i = 0; i < 3; i++) {
    for(int j = 0; j < 3; j++) {
        resultado[i][j] = m1[i][j] + m2[i][j];
    }
}
```

Recapitulando
Vetores e matrizes em C são estruturas de dados fundamentais para armazenar coleções de elementos do mesmo tipo que envolvem o uso de índices e, frequentemente, de loops para percorrer seus elementos.
Vetores são unidimensionais e acessados por um único índice, enquanto matrizes são bidimensionais e acessadas por dois índices. Matrizes são especialmente úteis para representar tabelas de dados, imagens e outras estruturas bidimensionais.

Sugestões de prática

- Escreva um programa em C que leia um vetor de 30 números inteiros e gere um segundo vetor cujos conteúdos das posições pares são o dobro do conteúdo do vetor original e os conteúdos das posições ímpares são o triplo do vetor original.
- Escreva um programa em C que leia um vetor A com 10 números inteiros, calcule e mostre a soma dos quadrados dos elementos do vetor.
- Escreva um programa em C que receba do usuário um vetor com 20 valores inteiros e apresente o maior, o menor e suas respectivas posições em que os mesmos foram informados. Caso existam números iguais mostre a posição da primeira ocorrência.
- Escreva um programa em C para calcular a matriz transposta MT de uma matriz M com dimensões m x n.

- Faça um programa em C que preencha uma matriz 3x3 de inteiros e escreva: a) A soma dos números ímpares fornecidos; b) A soma de cada uma das 3 colunas; c) A soma de cada uma das 3 linhas.

- Escreva um programa em C que leia um conjunto de números inteiros e armazene em uma matriz A de ordem 5x5 de inteiros e depois manipule e mostre os dados na seguinte ordem: a) Mostre a matriz original; b) Troque a segunda linha da matriz com a quinta linha da matriz (modifique a matriz original); c) Troque a primeira coluna da matriz com a quarta coluna da matriz; d) Mostre a matriz modificada.

- Faça um programa em C que leia duas matrizes X e Y, quadradas com ordem n, e faça as seguintes manipulações: a) Gere uma matriz W com os menores elementos entre as matrizes X e Y, ou seja, um elemento Wij possui o menor valor entre os elementos Xij e Yij. Se os elementos forem iguais ambos são o menor valor. b) Gere um vetor com os elementos da diagonal principal da matriz X; c) Encontre a coluna da matriz Y que possui a menor soma do valor de seus elementos dentre todas as colunas da matriz, e mostre qual é a posição dessa coluna. Caso existam somas de colunas iguais considerar a primeira coluna com a ocorrência do valor;

9 Strings e funções

9.1 Strings

Em C, strings são vetores de caracteres terminados pelo caractere nulo '\0'. A linguagem oferece diversas funções da biblioteca string.h para manipulação de texto. A declaração pode ser feita de duas formas principais:

```
char nome[50] = "João";                 // Array de caracteres
char *texto = "Texto dinâmico";         // Ponteiro para string literal
```

As funções mais importantes para manipulação de strings incluem strlen(), strcpy(), strcat() e strcmp(). Aqui está um exemplo prático usando várias operações com strings:

```
#include <string.h>

char nome1[50] = "Programação";
char nome2[50] = " em C";
char resultado[100];

// Copiando strings
strcpy(resultado, nome1);           // resultado = "Programação"

// Concatenando
strcat(resultado, nome2);          // resultado = "Programação em C"

// Comprimento da string
int tamanho = strlen(resultado);   // tamanho = 15

// Comparação de strings
if(strcmp(nome1, nome2) == 0) {    // Retorna 0 se iguais
    printf("Strings iguais\n");
}
```

```
// Exemplo prático: função que inverte uma string
void inverterString(char str[]) {
    int len = strlen(str);
    for(int i = 0; i < len/2; i++) {
        char temp = str[i];
        str[i] = str[len-1-i];
        str[len-1-i] = temp;
    }
}
```

Em C, diferentemente de outras linguagens, não existe um tipo string nativo. Todas as operações são realizadas em arrays de caracteres e cabe ao programador deve sempre garantir espaço suficiente para o caractere nulo de terminação e gerenciar adequadamente o tamanho do buffer para evitar overflow.

10 Funções

10.1 O que é uma função?

Trata-se um *mini-programa* que executa uma tarefa específica. Imagine uma função como uma caixa que pode receber informações (parâmetros), faz algo com essas informações e pode devolver um resultado (retorno).

10.2 Estrutura básica

```
tipo_retorno nome_funcao(parametros) {
    // código da função
    return valor;  // se necessário
}

// Exemplo simples
int soma(int a, int b) {
    return a + b;
}
```

10.3 Tipos de funções

10.3.1 Função void (sem retorno)

```
void saudacao() {
    printf("Olá!\n");
}

// Como usar:
saudacao();  // Imprime: Olá!
```

10.3.2 Função com retorno

```
int dobro(int numero) {
    return numero * 2;
}

// Como usar:
int resultado = dobro(5);  // resultado = 10
```

10.3.3 Função com múltiplos parâmetros

```
float media(float n1, float n2) {
    return (n1 + n2) / 2;
}

// Como usar:
float m = media(7.5, 8.5);  // m = 8.0
```

Algumas dicas importantes acerca das funções são declará-las antes de usá-las, garantir que cada função faça apenas uma coisa e usar nomes claros, assim como ao nomear variáveis.

Recapitulando
Em C, strings são essencialmente sequências de caracteres terminadas por um caractere nulo ('\0'). Para manipulá-las, a linguagem oferece diversas funções, como `strcpy` para copiar, `strcat` para concatenar e `strlen` para obter o comprimento. É fundamental alocar memória suficiente para as strings e garantir que haja espaço para o caractere nulo. A biblioteca `string.h` fornece um conjunto completo de funções para trabalhar com strings.

Já as funções são blocos de código reutilizáveis que executam tarefas específicas. Elas recebem dados de entrada (parâmetros) e podem retornar um resultado. Ao definir uma função, é importante especificar o tipo de dado que ela retorna e os tipos dos parâmetros que ela recebe.

Sugestão de prática

- Faça um programa em C que receba uma frase, calcule e mostre a quantidade de vogais da frase digitada. O programa deverá considerar e contar as vogais maiúsculas e minúsculas.
- Escreva um programa em C que peça ao usuário para digitar um texto e um caractere (vogal ou consoante), sendo uma entrada por linha. A saída deve mostrar o número de vogais do texto digitado e imprimir um novo texto: as vogais do texto original devem ser substituídas pelo caractere digitado.
- Faça um programa em C que leia uma string e imprima uma mensagem dizendo se ela é um palíndromo ou não. Um palíndromo é uma palavra que tem a propriedade de poder ser lida tanto da direita para a esquerda como da esquerda para a direita. Exemplo: ovo, arara, rever, asa, osso etc.

Desafio

A Torre de Hanoi é um quebra-cabeça matemático que consiste em três hastes e um número de discos de diferentes tamanhos, colocados em uma das hastes em ordem decrescente de tamanho, formando uma torre. O objetivo é mover toda a torre para outra haste, seguindo estas regras: - Mover apenas um disco por vez. - Um disco maior nunca pode ser colocado sobre um disco menor. Sabendo disso, crie um programa em C que resolva o problema da Torre de Hanoi utilizando vetores para armazenar os discos em cada haste.

Dicas:

- **Represente as hastes:** Utilize três vetores para representar as três hastes. Cada posição do vetor representa um disco, e o valor armazenado indica o tamanho do disco (0 para uma posição vazia).
- **Implemente a função de movimentação:** Crie uma função que receba os índices das hastes origem, destino e auxiliar, e mova o disco superior da haste origem para a haste destino, verificando se a movimentação é válida.

- **Resolva o problema:** Utilize uma função recursiva para resolver o problema da Torre de Hanoi, similar à solução convencional, mas adaptando-a para utilizar os vetores.

11 Conclusão

Neste guia, você foi introduzido aos fundamentos da programação. Iniciamos com os algoritmos e a lógica de programação e em seguida exploramos conceitos básicos como tipos de dados, variáveis e operadores. Foram vistas as estruturas de controle para tomar decisões e repetir ações, assim como as estruturas de dados como vetores, matrizes e strings. Por fim, aprendemos a organizar o código em funções para maior modularidade e reutilização.

Este guia forneceu uma apenas base sucinta, e portanto é fundamental praticar para de fato aprender C.

Bons estudos!