

Domine a linguagem C

Tudo o que você precisa saber em um único lugar



[Fábio Souza](#)

2023

Aviso legal

Este eBook foi escrito com fins didáticos, e com todos os esforço para que ele ficasse o mais claro e didático possível.

O objetivo deste eBook é educar. O autor não garante que as informações contidas neste eBook estão totalmente completas e não deve ser responsável por quaisquer erros ou omissões.

O autor não será responsabilizado para com qualquer pessoa ou entidade com relação a qualquer perda, ou dano causado, ou alegado a ser causado direta, ou indiretamente por este eBook.

Se achar algum erro, ou tiver sugestões de tópicos, ou melhorias, me envie um e-mail: contato@embarcados.com.br



Este obra está licenciado com uma Licença [Creative Commons Atribuição-NãoComercial-CompartilhaIgual 4.0 Internacional](https://creativecommons.org/licenses/by-nc-sa/4.0/).

Sumario

Introdução	4
Variáveis	5
Tipos de dados	5
Operadores	7
Operadores aritméticos	7
Operadores relacionais	9
Operadores lógicos	11
Operadores bit-a-bit	12
Operadores especiais	14
sizeof()	14
Endereço (&) e ponteiro(*)	15
Operador Ternário	16
Estruturas de controle de fluxo	17
if, if-else	17
switch	19
for	21
while	22
do-while	23
Funções	25
Vetores e matrizes	27
Ponteiros	29
Structs	30
Strings	32
Constantes	35
Modificadores de tipo	36
Modificadores de Armazenamento	37
Diretivas de pré-compilação	40
Compilador GCC	41
Compilando um código com o GCC	41
Criando bibliotecas	42
Saiba Mais	45

Introdução

A linguagem C é uma linguagem de programação de alto nível que foi criada nos anos 1970 para desenvolvimento de sistemas operacionais. Ela é amplamente utilizada em sistemas embarcados.

A seguir é apresentado um resumo das principais estruturas da linguagem C. Serão apresentados as Estruturas Básicas, Tipos de Dados, Operadores, Estruturas de Controle de Fluxo, Funções, Vetores e Matrizes, Ponteiros e Structs.

Use esse guia sempre que tiver dúvidas sobre algum tópico da linguagem C. Ao final do texto você encontrará mais artigos e referências com tópicos avançados e aplicações da linguagem C em sistemas embarcados.

Dica: Para reproduzir os códigos exemplos apresentados, você pode usar a ferramenta: [OnlineGDB](#)

Variáveis

As variáveis são locais de armazenamento em memória onde os dados podem ser armazenados e manipulados durante a execução de um programa.

Aqui está um exemplo de como declarar e atribuir um valor a uma variável em C:

```
#include <stdio.h>

int main(void) {
    // Declara uma variável inteira chamada "idade"
    int idade;

    // Atribui o valor 25 à variável "idade"
    idade = 25;

    printf("Minha idade é: %d\n", idade);

    return 0;
}
```

Neste exemplo, a variável "idade" é declarada como um inteiro. Ela é então atribuída o valor 25 através da atribuição de valor "idade = 25". O valor da variável "idade" é então exibido na tela através da função printf().

O resultado da execução deste programa seria: "Minha idade é: 25".

Tipos de dados

A linguagem C possui vários tipos de dados, como inteiros, ponto flutuante, caracteres e booleanos.

Veja como usar diferentes tipos de dados em C:

```
#include <stdio.h>

int main(void) {
    // Declara uma variável inteira chamada "idade" e atribui o
    // valor 25
    int idade = 25;

    // Declara uma variável de ponto flutuante chamada "altura" e
    // atribui o valor 1.80
    float altura = 1.80;

    // Declara uma variável de caractere chamada "sexo" e atribui o
    // valor 'M'
    char sexo = 'M';

    // Declara uma variável booleana chamada "casado" e atribui o
    // valor verdadeiro
    _Bool casado = 1;

    printf("Idade: %d\nAltura: %f\nSexo: %c\nCasado: %d\n", idade,
    altura, sexo, casado);

    return 0;
}
```

Neste exemplo, são declaradas quatro variáveis de diferentes tipos: inteiro, ponto flutuante, caractere e booleano. Cada uma delas é atribuída um valor apropriado e, em seguida, é exibida na tela através da função `printf()`.

O resultado da execução deste programa:

```
Idade: 25
Altura: 1.800000
Sexo: M
Casado: 1

...Program finished with exit code 0
Press ENTER to exit console.█
```

Operadores

Os operadores são símbolos especiais utilizados para realizar operações matemáticas e lógicas.

Operadores aritméticos

Os operadores aritméticos são usados para realizar operações matemáticas básicas em C, como adição, subtração, multiplicação e divisão.

Operador	Operação
+	soma
-	subtração
*	multiplicação
/	divisão inteira
%	módulo
++	incremento
--	decremento

```
#include <stdio.h>

int main(void) {
    int a = 5, b = 3;

    int soma = a + b;
    printf("soma de %d e %d é %d\n", a, b, soma);

    int subtracao = a - b;
    printf("subtracao de %d e %d é %d\n", a, b, subtracao);

    int mult = a * b;
    printf("multiplicação de %d e %d é %d\n", a, b, mult);

    int divisao = a / b;
    printf("divisao de %d e %d é %d\n", a, b, divisao);

    int mod = a % b;
    printf("resto da divisão de %d e %d é %d\n", a, b, mod);

    a++;
    printf("incremento: %d\n", a);

    b--;
    printf("decremento: %d\n", b);

    return 0;
}
```


Resultado:

```
soma de 5 e 3 é 8
subtracao de 5 e 3 é 2
multiplicação de 5 e 3 é 15
divisao de 5 e 3 é 1
resto da divisão de 5 e 3 é 2
incremento: 6
decremento: 2

...Program finished with exit code 0
Press ENTER to exit console. □
```

Operadores relacionais

Os operadores relacionais são usados para comparar dois valores em C e produzir um valor lógico (verdadeiro(1) ou falso(0)) como resultado.

Operador	Operação
==	igual
!=	diferente
>	maior do que
<	menor do que
>=	maior ou igual
<=	menor ou igual

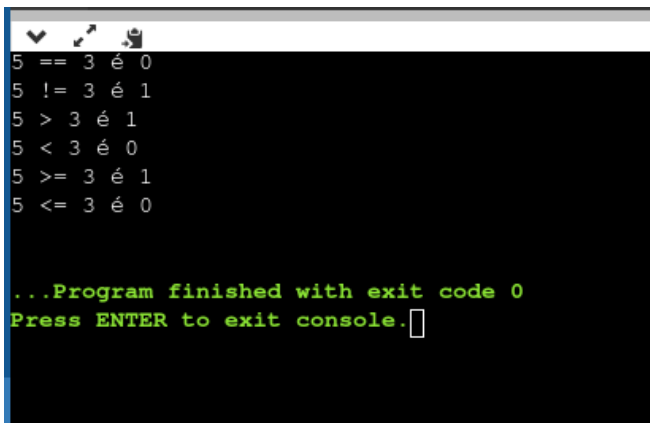
Exemplo:

```
#include <stdio.h>

int main(void) {
    int a = 5, b = 3;
    //Igualdade
    int resultado = (a == b);
    printf("%d == %d é %d\n", a, b, resultado);
    //Diferença
    resultado = (a != b);
    printf("%d != %d é %d\n", a, b, resultado);
    //Maior
    resultado = (a > b);
    printf("%d > %d é %d\n", a, b, resultado);
    //Menor
    resultado = (a < b);
    printf("%d < %d é %d\n", a, b, resultado);
    //Maior ou igual
    resultado = (a >= b);
    printf("%d >= %d é %d\n", a, b, resultado);
    //Menor ou Igual
    resultado = (a <= b);
    printf("%d <= %d é %d\n", a, b, resultado);

    return 0;
}
```

Resultado



```
5 == 3 é 0
5 != 3 é 1
5 > 3 é 1
5 < 3 é 0
5 >= 3 é 1
5 <= 3 é 0

...Program finished with exit code 0
Press ENTER to exit console. □
```

Operadores lógicos

Os operadores lógicos são usados para combinar expressões booleanas em C e produzir um valor lógico (verdadeiro ou falso) como resultado.

Operador	Operação
&&	AND (E)
	OR (OU)
!	NOT (NÃO)

Exemplo:

```
#include <stdio.h>

int main(void) {
    int a = 1, b = 1;

    //AND (E)
    int resultado = (a && b);
    printf("%d && %d é %d\n", a, b, resultado);
    //OR (OU)
    resultado = (a || b);
    printf("%d || %d é %d\n", a, b, resultado);
    //NOT (NÃO)
    resultado = !a;
    printf("!%d é %d\n", a, resultado);
    //Combinação de operadores lógicos
    resultado = (a && b) || !a;
    printf("(%d && %d) || !%d é %d\n", a, b, a, resultado);

    return 0;
}
```

Resultado:

```

1 && 1 é 1
1 || 1 é 1
!1 é 0
(1 && 1) || !1 é 1

...Program finished with exit code 0
Press ENTER to exit console.

```

Operadores bit-a-bit

Os operadores bit-a-bit são usados para realizar operações bit-a-bit em C. Eles são úteis quando você precisa trabalhar com números binários diretamente.

Operador	Operação
&	AND (E)
	OR (OU)
^	XOR (OU-EXCLUSIVO)
~	NOT (NÃO)
<<	deslocamento de bits à esquerda
>>	deslocamento de bits à direita

Exemplo:

```
#include <stdio.h>

int main(void) {
    int a = 5, b = 3;

    //AND
    int resultado = (a & b);
    printf("%d & %d é %d\n", a, b, resultado);
    //OR
    resultado = (a | b);
    printf("%d | %d é %d\n", a, b, resultado);
    //XOR
    resultado = (a ^ b);
    printf("%d ^ %d é %d\n", a, b, resultado);
    //NOT
    resultado = (~a);
    printf("~%d é %d\n", a, resultado);
    //deslocamento de bits à esquerda
    resultado = (a << 1);
    printf("%d << 1 é %d\n", a, resultado);
    //deslocamento de bits à direita
    resultado = (a >> 1);
    printf("%d >> 1 é %d\n", a, resultado);

    return 0;
}
```

Resultado:

```

5 & 3 é 1
5 | 3 é 7
5 ^ 3 é 6
~5 é -6
5 << 1 é 10
5 >> 1 é 2

...Program finished with exit code 0
Press ENTER to exit console.

```

Operadores especiais

sizeof()

O operador sizeof() é usado para descobrir o tamanho, em bytes, de uma variável ou tipo de dados em C. Ele é útil para determinar o espaço de armazenamento necessário para uma determinada variável ou estrutura.

```

#include <stdio.h>

int main(void) {
    int a = 5;
    printf("O tamanho de a é %lu bytes\n", sizeof(a));

    double b = 3.14;
    printf("O tamanho de b é %lu bytes\n", sizeof(b));

    char c = 'x';
    printf("O tamanho de c é %lu bytes\n", sizeof(c));

    return 0;
}

```

```
0 tamanho de a é 4 bytes
0 tamanho de b é 8 bytes
0 tamanho de c é 1 bytes

...Program finished with exit code 0
Press ENTER to exit console. □
```

Endereço (&) e ponteiro(*)

O operador de endereço (&) é usado para obter o endereço de uma variável em C, enquanto os ponteiros(*) são variáveis que armazenam esses endereços. Eles permitem que você acesse e altere o valor de uma variável através de seu endereço.

```
#include <stdio.h>

int main(void) {
    int a = 5;
    int* p = &a;

    printf("O valor de a é %d\n", a);
    printf("O endereço de a é %p\n", (void*)&a);
    printf("O valor armazenado no ponteiro p é %p\n", (void*)p);
    printf("O valor acessado através do ponteiro p é %d\n", *p);

    *p = 7;
    printf("Novo valor de a é %d\n", a);

    return 0;
}
```

```

O valor de a é 5
O endereço de a é 0x7ffffdc8586fc
O valor armazenado no ponteiro p é 0x7ffffdc8586fc
O valor acessado através do ponteiro p é 5
Novo valor de a é 7

...Program finished with exit code 0
Press ENTER to exit console.

```

Operador Ternário

O operador ternário é um operador condicional que permite escrever expressões condicionais de maneira concisa e legível. Ele é composto por três partes: uma expressão condicional, seguida de um ponto de interrogação (?), seguida de uma expressão a ser avaliada caso a expressão condicional seja verdadeira, e um dois-pontos (:), seguido de uma expressão a ser avaliada caso a expressão condicional seja falsa. A sintaxe geral é:

```
condição ? expressão1 : expressão2
```

Aqui está um exemplo de como usar o operador ternário em C:

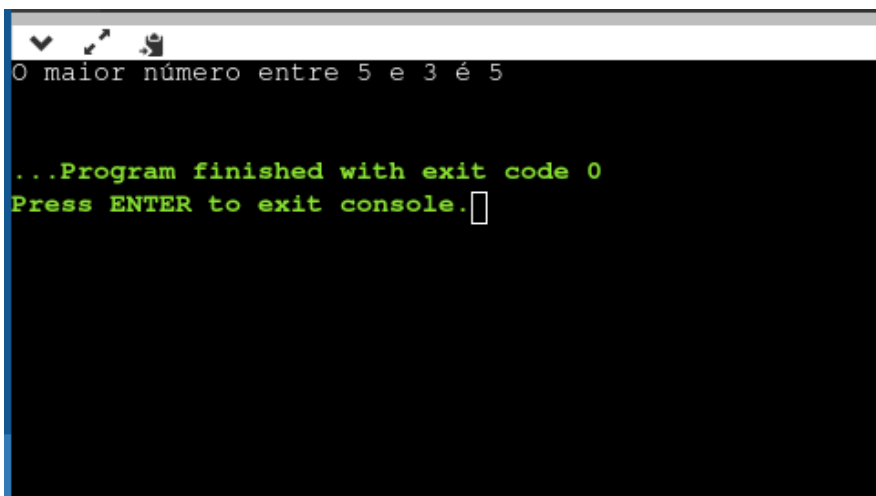
```
#include <stdio.h>

int main(void) {
    int a = 5, b = 3;
    int maior = (a > b) ? a : b;
    printf("O maior número entre %d e %d é %d\n", a, b, maior);

    return 0;
}
```


Neste exemplo, a expressão condicional "(a > b)" é avaliada, e caso seja verdadeira, a expressão "a" é avaliada e armazenada na variável 'maior', caso contrário, a expressão "b" é avaliada e armazenada na variável 'maior'.

O operador ternário é uma maneira concisa de escrever expressões condicionais e pode ser usado para simplificar muitas estruturas de decisão condicionais comuns. Ele permite expressar de forma concisa e limpa a lógica de uma operação de condicional simples, e é bastante útil para expressar operações condicionais simples.



```
0 maior número entre 5 e 3 é 5

...Program finished with exit code 0
Press ENTER to exit console. █
```

Estruturas de controle de fluxo

As estruturas de controle de fluxo são usadas para controlar a execução de um programa, permitindo que ele execute diferentes ações de acordo com determinadas condições.

if, if-else

A instrução "if" é uma estrutura de controle de fluxo que permite que você execute diferentes trechos de código com base em uma determinada condição. A sintaxe geral é:

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

Você pode incluir uma instrução "else" opcional para especificar o código a ser executado se a condição for falsa:

```

if (condição) {
    // código a ser executado se a condição for verdadeira
} else {
    // código a ser executado se a condição for falsa
}

```

Aqui está um exemplo de como usar a instrução "if" em C:

```

#include <stdio.h>

int main(void) {
    int idade = 30;
    if (idade < 18) {
        printf("Você é menor de idade.\n");
    } else {
        printf("Você é maior de idade.\n");
    }
    return 0;
}

```

Neste exemplo, a condição "idade < 18" é avaliada, e se for verdadeira, o código dentro do primeiro bloco de chaves é executado e imprime "Você é menor de idade.", senão, o código dentro do segundo bloco é executado e imprime "Você é maior de idade."

Além disso, você pode adicionar mais de um "if" e "else" para criar múltiplas condições ou incluir uma estrutura "else if" para adicionar mais condições e evitar a necessidade de aninhar vários ifs.

```

if (idade < 18)
{
    printf("Você é menor de idade.\n");
}
else if (idade >= 18 && idade <= 30)
{
    printf("Você é jovem adulto.\n");
}
else
{
    printf("Você é adulto.\n");
}

```

switch

A instrução "switch" é uma estrutura de controle de fluxo que permite que você execute diferentes trechos de código com base em um valor de expressão. A sintaxe geral 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 valores acima for  
        igual a expressão  
}
```

A expressão é avaliada e comparada com cada um dos valores "case". Quando um valor é encontrado que é igual à expressão, o código dentro do bloco correspondente é executado. A palavra-chave "break" é usada para sair do bloco e continuar a execução do código após o switch.

Se nenhum dos valores "case" for igual à expressão, o código dentro do bloco "default" será executado, se houver. Se não houver um bloco "default", o código após o switch será executado sem entrar em nenhum dos case.

Aqui está um exemplo de como usar a instrução "switch" em C:

```
#include <stdio.h>

int main(void) {
    char opcao = 'B';
    switch (opcao) {
        case 'A':
            printf("Opção A escolhida.\n");
            break;
        case 'B':
            printf("Opção B escolhida.\n");
            break;
        case 'C':
            printf("Opção C escolhida.\n");
            break;
        default:
            printf("Opção inválida.\n");
    }
    return 0;
}
```

Neste exemplo, a expressão "opcao" é avaliada e comparada com cada um dos valores "case". Se a expressão for igual a 'A', 'B', ou 'C', a mensagem de "Opção A escolhida.", "Opção B escolhida.", ou "Opção C escolhida." será impressa respectivamente. Se não for igual a nenhum dos valores acima, "Opção inválida." será impressa.

for

A instrução "for" é uma estrutura de controle de fluxo que permite que você execute um trecho de código repetidamente enquanto uma determinada condição for verdadeira. A sintaxe geral é:

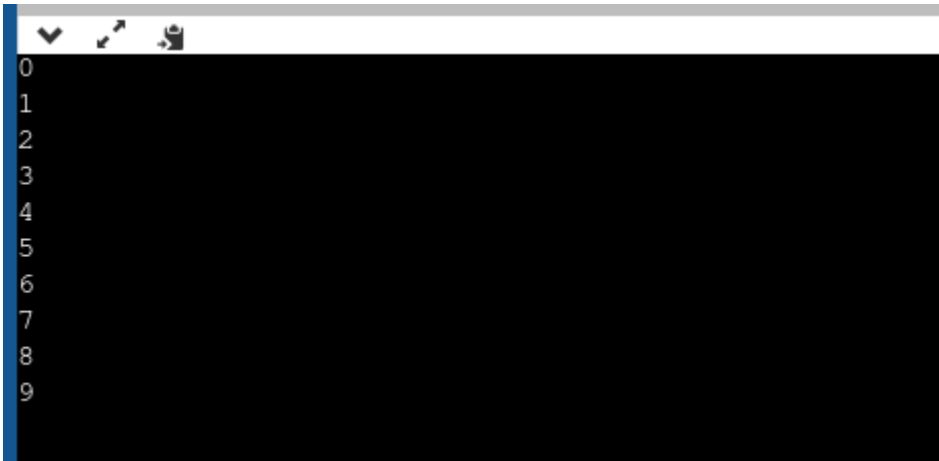
```
for (inicialização; condição; incremento) {  
    // código a ser executado enquanto a condição for verdadeira  
}
```

A inicialização é executada uma vez no início do loop, a condição é verificada antes de cada iteração, e o incremento é executado após cada iteração. Se a condição for verdadeira, o código dentro do bloco será executado. Se a condição for falsa, o loop é interrompido e a execução continua após o loop.

Aqui está um exemplo de como usar a instrução "for" em C:

```
#include <stdio.h>  
  
int main(void) {  
    for (int i = 0; i < 10; i++) {  
        printf("%d\n", i);  
    }  
    return 0;  
}
```

Neste exemplo, a variável "i" é inicializada com o valor 0, a condição "i < 10" é verificada antes de cada iteração, e o incremento "i++" é executado após cada iteração. O código dentro do loop imprime o valor de "i" em cada iteração, e o loop é executado 10 vezes, até que a condição "i < 10" se torne falsa.

A screenshot of a terminal window with a black background and a blue border on the left. The terminal shows a loop that prints the numbers 0 through 9, one per line. The numbers are displayed in white text. The terminal window has standard icons at the top: a checkmark, a refresh symbol, and a trash can.

while

A instrução "while" é uma estrutura de controle de fluxo que permite que você execute um trecho de código repetidamente enquanto uma determinada condição for verdadeira. A sintaxe geral é:

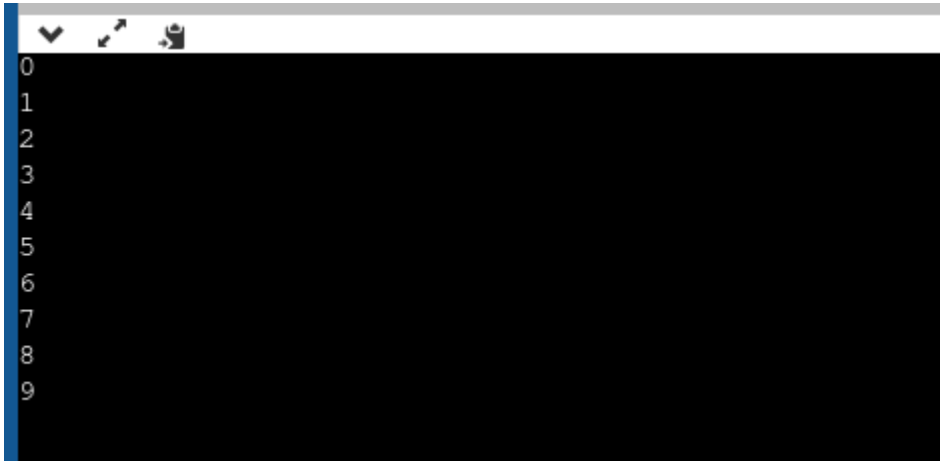
```
while (condição) {  
    // código a ser executado enquanto a condição for verdadeira  
}
```

A condição é verificada antes de cada iteração. Se a condição for verdadeira, o código dentro do bloco será executado. Se a condição for falsa, o loop é interrompido e a execução continua após o loop.

Aqui está um exemplo de como usar a instrução "while" em C:

```
#include <stdio.h>  
  
int main(void) {  
    int i = 0;  
    while (i < 10) {  
        printf("%d\n", i);  
        i++;  
    }  
    return 0;  
}
```

Neste exemplo, a variável "i" é inicializada com o valor 0, e a condição "i < 10" é verificada antes de cada iteração. O código dentro do loop imprime o valor de "i" em cada iteração, e o incremento "i++" é executado após cada iteração, e o loop é executado 10 vezes, até que a condição "i < 10" se torne falsa.

A terminal window with a black background and a blue border on the left. The window displays the numbers 0 through 9, one on each line, representing the output of a loop. At the top of the terminal, there are three small icons: a downward arrow, a double arrow, and a trash can icon.

do-while

A instrução "do-while" é uma estrutura de controle de fluxo semelhante à instrução "while", mas com uma diferença importante: o código dentro do loop é executado pelo menos uma vez, independentemente da condição. A sintaxe geral é:

```
do {  
    // código a ser executado  
} while (condição);
```

A condição é verificada após cada iteração. Se a condição for verdadeira, o código dentro do bloco será executado novamente. Se a condição for falsa, o loop é interrompido e a execução continua após o loop.

Aqui está um exemplo de como usar a instrução "do-while" em C:

```
#include <stdio.h>

int main(void) {
    int i = 0;
    do {
        printf("%d\n", i);
        i++;
    } while (i < 10);
    return 0;
}
```

Neste exemplo, a variável "i" é inicializada com o valor 0, e o código dentro do loop imprime o valor de "i" em cada iteração, e o incremento "i++" é executado após cada iteração. A condição "i < 10" é verificada após cada iteração e o loop é executado até 10 vezes, até que a condição se torne falsa.

A nota que é importante que a variável "i" seja atualizada dentro do loop, para que ele não execute infinitamente.

O laço "do-while" é útil para situações onde você precisa garantir que o código dentro do loop seja executado pelo menos uma vez, independentemente da condição.

Funções

As funções são blocos de código que podem ser reutilizados em um programa. Elas permitem que o código seja dividido em partes menores e mais fáceis de entender e manter.

Exemplo:

```
#include <stdio.h>

// Declara uma função chamada "soma" que recebe dois inteiros como
// argumentos e retorna um inteiro
int soma(int x, int y) {
    // Calcula a soma de x e y e retorna o resultado
    return x + y;
}

int main(void) {
    // Declara duas variáveis inteiras e atribui valores
    int x = 5;
    int y = 2;

    // Chama a função "soma" e armazena o resultado em uma variável
    int resultado = soma(x, y);

    // Exibe o resultado da soma
    printf("%d + %d = %d\n", x, y, resultado);

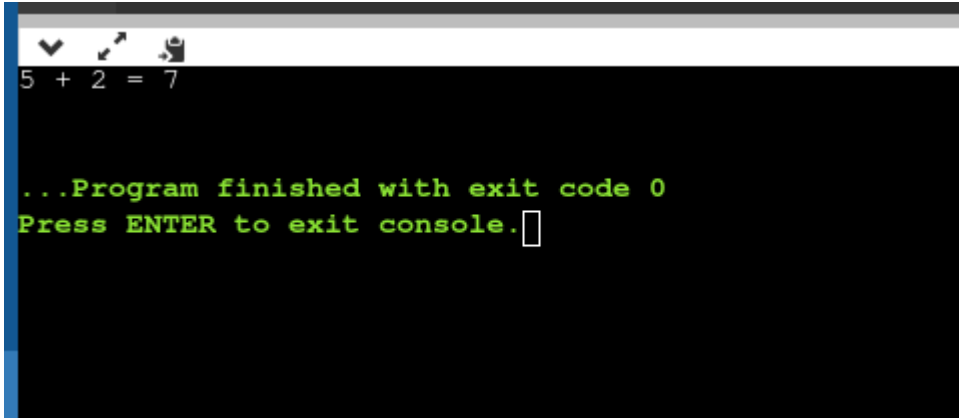
    return 0;
}
```

Neste exemplo, é criada uma função chamada "soma" que recebe dois inteiros como argumentos e retorna um inteiro. A função "soma" é definida logo abaixo da declaração da função. Ela realiza a soma dos dois argumentos e retorna o resultado.

Na função main(), as variáveis "x" e "y" são declaradas e atribuídas os valores 5 e 2, respectivamente. Em seguida, a função "soma" é chamada com os argumentos "x" e "y" e o resultado é armazenado na

variável "resultado". Por fim, o resultado da soma é exibido na tela através da função printf().

O resultado da execução deste programa seria: "5 + 2 = 7"



```
5 + 2 = 7

...Program finished with exit code 0
Press ENTER to exit console. █
```

Vetores e matrizes

Os vetores são conjuntos de dados do mesmo tipo armazenados em sequência em memória. As matrizes são vetores multidimensionais que permitem armazenar mais de um valor em cada posição.

```
#include <stdio.h>

#define TAMANHO 10

int main(void) {
    // Declara um vetor de inteiros com 10 elementos
    int vetor[TAMANHO] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};

    // Exibe o conteúdo do vetor
    printf("Conteúdo do vetor:\n");
    for (int i = 0; i < TAMANHO; i++) {
        printf("%d ", vetor[i]);
    }
    printf("\n");

    // Declara uma matriz de inteiros com 3 linhas e 2 colunas
    int matriz[3][2] = {
        {1, 2},
        {3, 4},
        {5, 6}
    };

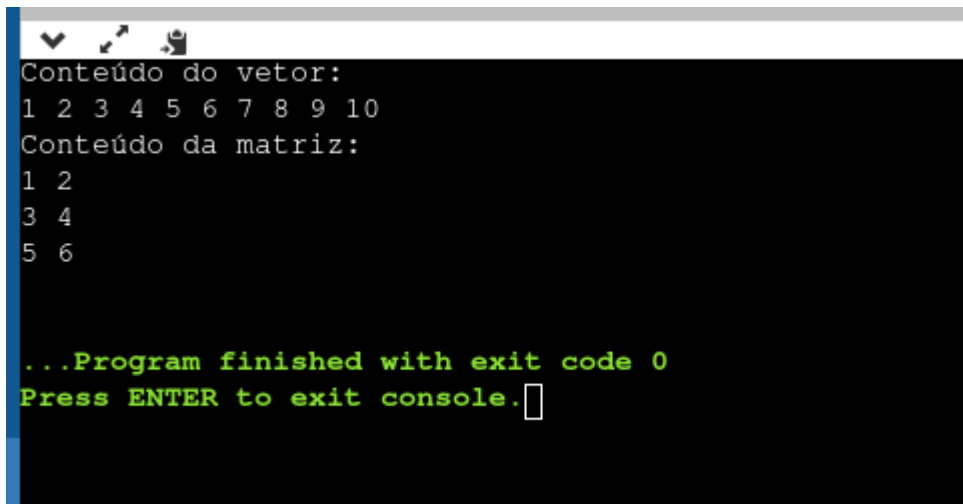
    // Exibe o conteúdo da matriz
    printf("Conteúdo da matriz:\n");
    for (int i = 0; i < 3; i++) {
        for (int j = 0; j < 2; j++) {
            printf("%d ", matriz[i][j]);
        }
        printf("\n");
    }

    return 0;
}
```

Neste exemplo, um vetor de inteiros com 10 elementos é declarado e inicializado com valores. Em seguida, um loop "for" é usado para exibir o conteúdo do vetor na tela.

Também é declarada uma matriz de inteiros com 3 linhas e 2 colunas e inicializada com valores. Em seguida, um loop aninhado é usado para exibir o conteúdo da matriz na tela.

O resultado da execução deste programa seria: "Conteúdo do vetor: 1 2 3 4 5 6 7 8 9 10 Conteúdo da matriz: 1 2 3 4 5 6"



```
Conteúdo do vetor:
1 2 3 4 5 6 7 8 9 10
Conteúdo da matriz:
1 2
3 4
5 6

...Program finished with exit code 0
Press ENTER to exit console.
```

Ponteiros

Os ponteiros são variáveis que armazenam endereços de memória de outras variáveis. Eles são amplamente utilizados em C para acessar e manipular dados em memória de maneira mais eficiente.

Exemplo

```
#include <stdio.h>

int main(void) {
    // Declara uma variável inteira chamada "x" e atribui o valor 5
    int x = 5;

    // Declara um ponteiro para um inteiro chamado "p" e atribui o
    endereço de memória de "x"
    int *p = &x;

    // Exibe o valor de "x" e o endereço de memória de "x"
    printf("x = %d\nEndereço de x: %p\n", x, &x);

    // Exibe o valor apontado pelo ponteiro "p" e o endereço de
    memória armazenado em "p"
    printf("*p = %d\nEndereço armazenado em p: %p\n", *p, p);

    // Modifica o valor apontado pelo ponteiro "p"
    *p = 10;

    // Exibe o novo valor de "x"
    printf("Novo valor de x: %d\n", x);

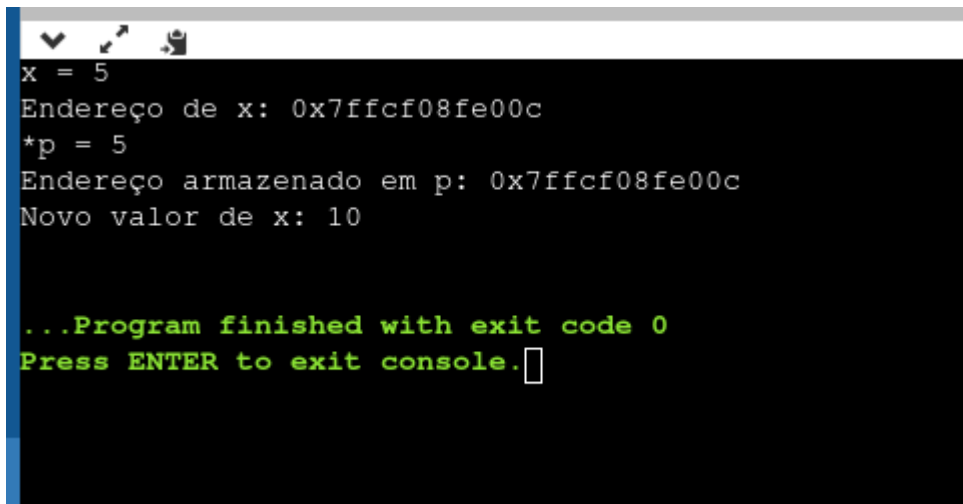
    return 0;
}
```

Neste exemplo, é declarada uma variável inteira chamada "x" e atribuído o valor 5. Em seguida, é declarado um ponteiro para um inteiro chamado "p" e atribuído o endereço de memória de "x".

O valor de "x" e o endereço de memória de "x" são exibidos na tela através da função printf(). Em seguida, o valor apontado pelo ponteiro "p" e o endereço de memória armazenado em "p" são exibidos na tela.

O valor apontado pelo ponteiro "p" é então modificado para 10 e o novo valor de "x" é exibido na tela.

O resultado da execução deste programa seria: "x = 5 Endereço de x: 0x7ffc7a70a260 *p = 5 Endereço armazenado em p: 0x7ffc7a70a260 Novo valor de x: 10"



```
x = 5
Endereço de x: 0x7ffcf08fe00c
*p = 5
Endereço armazenado em p: 0x7ffcf08fe00c
Novo valor de x: 10

...Program finished with exit code 0
Press ENTER to exit console.█
```

Structs

As "structs" (estruturas) são uma forma de agrupar diferentes tipos de dados em um único bloco de memória. Elas são muito úteis quando precisamos armazenar e gerenciar informações complexas, como registros de banco de dados ou informações de usuários em um sistema.

Por exemplo, imagine que você precise armazenar informações de um conjunto de pessoas, como nome, idade e sexo. Em vez de criar três variáveis diferentes para cada pessoa, você pode criar uma struct chamada "Pessoa" com esses campos e armazenar todas as informações em uma única variável do tipo "Pessoa".

A sintaxe para criar uma struct em C é a seguinte:

```
struct NomeDaStruct {
    tipo_de_dado campo1;
    tipo_de_dado campo2;
    tipo_de_dado campo3;
    // ...
};
```

Exemplo:

```
#include <stdio.h>
#include <string.h>

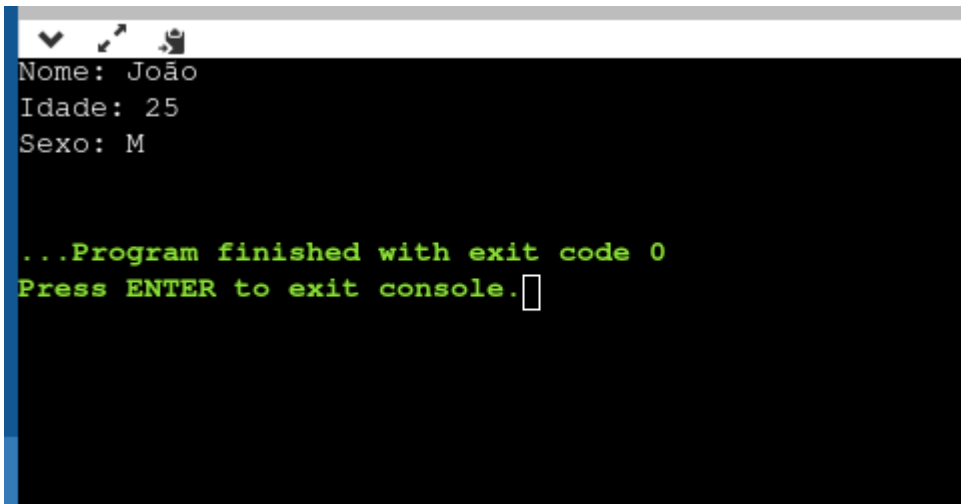
// Declara uma struct chamada "Pessoa" com os campos "nome",
"idade" e "sexo"
struct Pessoa {
    char nome[50];
    int idade;
    char sexo;
};

int main(void) {
    // Declara uma variável do tipo "Pessoa" chamada "pessoa1"
    struct Pessoa pessoa1;

    // Atribui valores aos campos da struct "pessoa1"
    strcpy(pessoa1.nome, "João");
    pessoa1.idade = 25;
    pessoa1.sexo = 'M';

    // Exibe os valores dos campos da struct "pessoa1"
    printf("Nome: %s\nIdade: %d\nSexo: %c\n", pessoa1.nome,
pessoa1.idade, pessoa1.sexo);

    return 0;
}
```

A screenshot of a terminal window with a black background and white text. The text shows the output of a C program: 'Nome: João', 'Idade: 25', and 'Sexo: M'. Below this, there is a green message: '...Program finished with exit code 0' and 'Press ENTER to exit console.' followed by a white cursor box.

```
Nome: João
Idade: 25
Sexo: M

...Program finished with exit code 0
Press ENTER to exit console.█
```

Strings

Em C, uma string é uma sequência de caracteres. Ela é representada por um vetor de caracteres, onde o último elemento é um caractere nulo ('\0').

Por exemplo, a string "Olá, mundo!" pode ser representada como um vetor de caracteres da seguinte forma:

```
char string[] = {'O', 'l', 'á', ',', ' ', 'm', 'u', 'n', 'd', 'o', '!', '\0'};
```

Para trabalhar com strings em C, existem várias funções disponíveis na biblioteca "string.h". Algumas das funções mais comuns são:

- `strlen()`: retorna o tamanho de uma string (sem contar o caractere nulo).
- `strcpy()`: copia uma string para outra.
- `strcat()`: concatena duas strings.
- `strcmp()`: compara duas strings.

Aqui está um exemplo de como usar algumas dessas funções:

```
#include <stdio.h>
#include <string.h>

int main(void) {
    // Declara duas strings
    char string1[] = "Olá, mundo!";
    char string2[20];

    // Copia a string "string1" para "string2"
    strcpy(string2, string1);

    // Exibe as duas strings
    printf("string1: %s\n", string1);
    printf("string2: %s\n", string2);

    // Exibe o tamanho das strings
    printf("Tamanho de string1: %ld\n", strlen(string1));
    printf("Tamanho de string2: %ld\n", strlen(string2));

    // Concatena as strings e armazena o resultado em uma terceira
string
    char string3[20];
    strcpy(string3, string1);
    strcat(string3, string2);

    // Exibe a string resultante da concatenação
    printf("string3: %s\n", string3);

    // Compara as strings "string1" e "string2"
    if (strcmp(string1, string2) == 0) {
        printf("As strings são iguais.\n");
    } else {
        printf("As strings são diferentes.\n");
    }

    return 0;
}
```

```
string1: Olá, mundo!  
string2: Olá, mundo!  
Tamanho de string1: 12  
Tamanho de string2: 12  
string3: Olá, mundo!Olá, mundo!  
As strings são iguais.  
  
...Program finished with exit code 0  
Press ENTER to exit console. □
```

Exemplo: Converter uma string para maiúsculas ou minúsculas:

```
#include <stdio.h>  
#include <ctype.h>  
  
int main(void) {  
    // Declara uma string  
    char string[] = "embarcados!";  
  
    // Converte a string para maiúsculas  
    for (int i = 0; string[i] != '\0'; i++) {  
        string[i] = toupper(string[i]);  
    }  
    printf("String em maiúsculas: %s\n", string);  
  
    // Converte a string para minúsculas  
    for (int i = 0; string[i] != '\0'; i++) {  
        string[i] = tolower(string[i]);  
    }  
    printf("String em minúsculas: %s\n", string);  
  
    return 0;  
}
```

```
String em maiúsculas: EMBARCADOS!  
String em minúsculas: embarcados!  
  
...Program finished with exit code 0  
Press ENTER to exit console. □
```

Constantes

Em C, as constantes são valores que não podem ser modificados durante a execução do programa. Elas são declaradas com a palavra-chave "const" e podem ser do tipo inteiro, flutuante, caractere ou string.

Aqui estão alguns exemplos de como usar constantes em C:

Declarar uma constante inteira:

```
#include <stdio.h>  
  
int main(void) {  
    // Declara uma constante inteira chamada "MAX_VALOR" e atribui o  
    // valor 10  
    const int MAX_VALOR = 10;  
  
    // Exibe o valor da constante  
    printf("MAX_VALOR = %d\n", MAX_VALOR);  
  
    // Tentativa de modificar o valor da constante (gerará um erro  
    // de compilação)  
    // MAX_VALOR = 20;  
  
    return 0;  
}
```

Declarar uma constante flutuante:

```
#include <stdio.h>

int main(void) {
    // Declara uma constante flutuante chamada "PI" e atribui o
    // valor 3.14
    const float PI = 3.14;

    // Exibe o valor da constante
    printf("PI = %f\n", PI);

    // Tentativa de modificar o valor da constante (gerará um erro
    // de compilação)
    // PI = 3.141;

    return 0;
}
```

Modificadores de tipo

Os modificadores de tipo são palavras-chave usadas para modificar o tipo de uma variável ou função em C. Eles permitem especificar propriedades adicionais do tipo, como o tamanho ou o escopo de uma variável.

Os modificadores de tipo mais comuns em C são:

- "short": indica que uma variável é do tipo inteiro com tamanho reduzido (geralmente, um short ocupa menos espaço na memória do que um inteiro normal).
- "long": indica que uma variável é do tipo inteiro com tamanho aumentado (geralmente, um long ocupa mais espaço na memória do que um inteiro normal).
- "signed": indica que uma variável é do tipo inteiro com sinal (pode armazenar valores positivos, negativos e zero).
- "unsigned": indica que uma variável é do tipo inteiro sem sinal (só pode armazenar valores positivos e zero).

- "float": indica que uma variável é do tipo ponto flutuante com precisão simples.
- "double": indica que uma variável é do tipo ponto flutuante com precisão dupla (maior que a de um float).

Aqui estão alguns exemplos de como usar modificadores de tipo em C:

```
#include <stdio.h>

int main(void) {
    // Declara uma variável "a" do tipo inteiro com tamanho reduzido
    short a = 32767;

    // Declara uma variável "b" do tipo inteiro com tamanho
    // aumentado
    long b = 123456789;

    // Declara uma variável "c" do tipo inteiro com sinal
    signed c = -123;

    // Declara uma variável "d" do tipo inteiro sem sinal
    unsigned d = 123;

    // Declara uma variável "e" do tipo ponto flutuante com precisão
    // simples
    float e = 3.14;

    // Declara uma variável "f" do tipo ponto flutuante com precisão
    // dupla
    double f = 3.14159265358979323846;

    return 0;
}
```

Modificadores de Armazenamento

Os modificadores de armazenamento são palavras-chave usadas para controlar a maneira como as variáveis são armazenadas em memória

em C. Eles permitem especificar o escopo e a duração de uma variável, além de sua visibilidade em relação a outras partes do código.

Os modificadores de armazenamento mais comuns em C são:

- "auto": indica que uma variável é armazenada na memória do stack e tem escopo de bloco (visível apenas dentro do bloco de código onde é declarada). Este é o modificador de armazenamento padrão em C.

```
#include <stdio.h>

void func() {
    auto int a = 5; // variavel a é automatica e tem escopo de
funcao
    printf("Valor de a: %d\n", a);
}

int main(void) {
    func();
    return 0;
}
```

- "static": indica que uma variável é armazenada na memória estática e tem escopo de arquivo (visível em todo o arquivo de código onde é declarada). Além disso, uma variável static mantém seu valor entre as chamadas da função onde é declarada.

```
#include <stdio.h>

void func() {
    static int a = 5; // variavel a é estatica e tem escopo de
funcao
    printf("Valor de a: %d\n", a);
    a++;
}

int main(void) {
    func();
    func();
    func();
}
```

```
    return 0;
}
```

- "extern": indica que uma variável é armazenada em outro lugar e tem escopo de arquivo (visível em todo o arquivo de código onde é declarada). A variável extern é usada para acessar uma variável global declarada em outro arquivo de código.

```
// Arquivo "main.c"
#include <stdio.h>

// Declara uma variável global "contador" sem modificador de
// armazenamento
int contador = 0;

int main(void) {
    // Incrementa o valor de "contador"
    contador++;
    printf("contador = %d\n", contador);

    return 0;
}

// Arquivo "outro_arquivo.c"
#include <stdio.h>

extern int contador; // Declara a variável "contador" como externa

int funcao(void) {
    // Incrementa o valor de "contador"
    contador++;
    printf("contador = %d\n", contador);

    return 0;
}
```

- **register**: é usado para indicar ao compilador que uma variável deve ser armazenada em um registrador, em vez de em uma área de memória, se possível. Isso pode melhorar o desempenho do seu programa, pois acessar um registrador é geralmente mais rápido do que acessar a memória.

```
#include <stdio.h>

int main(void) {
    register int a = 5; // a é armazenada em um registrador, se
    possível
    printf("Valor de a: %d\n", a);
    return 0;
}
```

Diretivas de pré-compilação

As diretivas de pré-compilação são instruções especiais inseridas no código-fonte de um programa em C processadas pelo pré-processador antes da compilação propriamente dita. Elas são usadas para incluir arquivos de cabeçalho, definir constantes e macros, e controlar a inclusão ou exclusão de trechos de código baseados em condições pré-definidas.

As diretivas de pré-compilação são identificadas pelo símbolo "#" no início da linha. Algumas das diretivas mais comuns são:

"#include": inclui um arquivo de cabeçalho no código-fonte. Por exemplo:

```
#include <stdio.h>
```

"#define": define uma constante ou macro. Por exemplo:

```
#define PI 3.14
#define MAX(a, b) ((a) > (b) ? (a) : (b))
```


"#ifdef"/"#ifndef": controla a inclusão de trechos de código baseados em se uma determinada constante ou macro foi definida. Por exemplo:

```
#ifdef DEBUG
    printf("Modo de depuração ativado!\n");
#endif
```

"#if"/"#elif"/"#else": controla a inclusão de trechos de código baseados em expressões de teste. Por exemplo:

```
#if defined(DEBUG) || defined(TESTING)
    printf("Modo de depuração ou teste ativado!\n");
#elif defined(RELEASE)
    printf("Modo de lançamento ativado!\n");
#else
    printf("Modo desconhecido!\n");
#endif
```

Compilador GCC

O GCC (GNU Compiler Collection) é uma ferramenta de linha de comando que permite compilar código-fonte escrito na linguagem C. Você pode usá-lo para compilar arquivos ".c" e transformá-los em arquivos executáveis.

Compilando um código com o GCC

Aqui está como você pode usar o GCC para compilar um arquivo de código fonte C:

1. Abra o terminal no seu computador.
2. Navegue até o diretório onde o arquivo de código fonte está localizado. Você pode usar os comandos "cd" e "ls" para navegar pelos diretórios e ver os arquivos disponíveis.

3. Digite o comando "gcc nome_do_arquivo.c -o nome_do_executavel" e pressione enter para compilar o código. Isso vai criar um arquivo executável com o nome especificado após o -o
4. Para rodar o código basta digitar ./nome_do_executavel e pressionar enter.

Exemplo:

```
gcc program.c -o program
```

Compila o arquivo "program.c" e cria um arquivo executável chamado "program"

```
./program
```

Executa o arquivo criado anteriormente.

Além disso, existem algumas opções adicionais que você pode passar para o GCC para controlar a forma como o código é compilado, como -g (para habilitar informações de depuração), -O (para habilitar otimizações de código) e -Wall (para habilitar mensagens de aviso adicionais). Para mais detalhes acesse: [Using the GNU Compiler Collection \(GCC\)](#)

Criando bibliotecas

Para criar uma biblioteca em C, você precisa criar um ou mais arquivos de cabeçalho (.h) que contenham as declarações de função e estruturas de dados que você quer expor para outros programas, e um ou mais arquivos de implementação (.c) que contenham a implementação dessas funções.

Aqui está um exemplo de como criar uma biblioteca simples que expõe uma função para calcular o fatorial de um número inteiro:

Arquivo de cabeçalho (biblioteca.h):

```
#ifndef BIBLIOTECA_H
#define BIBLIOTECA_H

int fatorial(int n);

#endif /* BIBLIOTECA_H */
```

Arquivo de implementação (biblioteca.c):

```
#include "biblioteca.h"

int fatorial(int n) {
    int resultado = 1;
    for (int i = 2; i <= n; i++) {
        resultado *= i;
    }
    return resultado;
}
```

Para usar essa biblioteca em outro programa, basta incluir o arquivo de cabeçalho e linkar o arquivo de implementação durante a compilação. Por exemplo:

```
#include <stdio.h>
#include "biblioteca.h"

int main(void) {
    int n = 5;
    printf("%d! = %d\n", n, fatorial(n));
    return 0;
}
```

Para compilar esse programa, você pode usar o seguinte comando:

```
gcc -o programa programa.c biblioteca.c
```

Observação

Para evitar que uma biblioteca em C seja incluída mais de uma vez durante a compilação, você pode usar uma diretiva de pré-processador especial chamada "pragma once". Essa diretiva indica ao pré-processador do compilador que a biblioteca só deve ser incluída uma vez, independentemente de quantas vezes ela é referenciada no código-fonte.

Para usar a diretiva "pragma once", basta adicioná-la no início do seu arquivo de cabeçalho (.h). Por exemplo:

```
#pragma once

/* resto do código da biblioteca aqui */
```

Outra opção é usar a diretiva "define" para definir uma macro específica no início do seu arquivo de cabeçalho e, em seguida, verificar se essa macro já foi definida antes de incluir o código da biblioteca. Por exemplo:

```
#ifndef MINHA_BIBLIOTECA_H
#define MINHA_BIBLIOTECA_H

/* resto do código da biblioteca aqui */

#endif /* MINHA_BIBLIOTECA_H */
```

Essa abordagem é um pouco mais verbosa, mas pode ser útil em casos em que você quer controlar a inclusão da biblioteca de maneira mais explícita.

Saiba Mais

[Bibliotecas Estáticas e DLL's em Linguagem C](#)

[Técnicas de Mapeamento de Memória em Linguagem C](#)

[Objetos em Linguagem C](#)

[Estilo de código – Boas práticas de programação em linguagem C](#)

[Modificadores de Armazenamento na Linguagem C](#)

[Modificadores de Acesso na Linguagem C](#)

[Bits em Linguagem C – Conceito e Aplicação](#)

Siga o Embarcados na Redes Sociais

<https://www.facebook.com/osembarcados/>

<https://www.instagram.com/portalembarcados/>

<https://www.youtube.com/embarcadostv/>

<https://www.linkedin.com/company/embarcados/>

<https://twitter.com/embarcados>