Funções em C

1. O que são Funções?

Em C, uma **função** é um bloco de código que realiza uma tarefa específica e pode ser chamado de qualquer lugar no programa. As funções permitem organizar o código em pequenas seções, tornando-o mais modular e fácil de entender.

```c
#include <stdio.h>

int soma(int a, int b) {
    return a + b;
}

int main() {
    int resultado = soma(5, 7);
    printf("Resultado: %d\n", resultado);
    return 0;
}
```

2. Vantagens da Modularização

- Reutilização de Código: Escreva uma vez, use quantas vezes precisar.
- Facilidade de Depuração: Cada função pode ser testada separadamente.
- Organização e Legibilidade: Código modular e mais fácil de ler.

3. Estrutura de uma Função

Uma função em C é definida com a seguinte estrutura:

```c
tipo_retorno nome_da_funcao(tipo1 arg1, tipo2 arg2, ...) {
    // Código que a função executa
    return valor; // Se houver um valor a ser retornado
}
```

Vamos entender cada parte dessa estrutura.

a) Tipo de Retorno: O **tipo de retorno** define o tipo de dado que a função devolve após ser executada. O tipo de retorno pode ser qualquer tipo de dado em C, como int, float, char, etc. Se a função não precisa retornar um valor, usamos o tipo void.

Exemplos:

- `int`: A função retorna um valor inteiro
- `float`: A função retorna um valor em ponto flutuante.
- `void`: A função não retorna nada.

Exemplo:

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

Neste caso, a função soma retorna um valor int, que é o resultado da soma entre a e b.

b) Nome da Função: O **nome da função** é um identificador que permite chamá-la no código. Ele deve seguir as regras de nomenclatura de variáveis em C (por exemplo, não começar com número, não conter espaços, não usar palavras reservadas).

Exemplo:

```
int calcularArea(int largura, int altura) {
    return largura * altura;
}
```

Aqui, `calcularArea` é o nome da função que calcula a área de um retângulo.

4. O que são argumentos e parâmetros?

Argumentos e parâmetros são os valores e as variáveis que passamos para as funções.

- **Parâmetros** são as variáveis que a função recebe e utiliza dentro de seu corpo. São definidos na declaração da função.
- **Argumentos** são os valores reais passados quando a função é chamada.

Exemplo:

```
int multiplicar(int x, int y) { // x e y são os parâmetros
    return x * y;
}

int main() {
    int resultado = multiplicar(3, 4); // 3 e 4 são os argumentos
    printf("Resultado: %d\n", resultado);
    return 0;
}
```

5. Passagem de Parâmetros

Em C, a passagem de argumentos para as funções é por valor. Isso significa que a função recebe uma cópia do valor do argumento. Portanto, alterações no valor do parâmetro dentro da função não afetam o valor da variável original fora da função.

Exemplo de Passagem por Valor:

```c
#include <stdio.h>

void alterarValor(int num) {
    num = 20; // Atribuição só altera o valor da cópia
}

int main() {
    int valor = 10;
    alterarValor(valor);
    printf("Valor após a função: %d\n", valor); // Valor permanece 10
    return 0;
}
```

Um meio para contornar a passagem por valor é por meio dos **ponteiros**, assunto que será estudado posteriormente.

6. Retorno de Funções

Quando uma função termina sua execução, ela pode retornar um valor ao ponto onde foi chamada. O tipo de dado do retorno deve coincidir com o tipo de retorno definido na função.

Para retornar um valor, usamos a palavra-chave return, seguida do valor ou da expressão que queremos devolver.

Exemplo de Função com Retorno:

```c
#include <stdio.h>

int dobrar(int n) {
    return n * 2; // Retorna o dobro do valor de n (valor inteiro já que o tipo da
função é int)
}

int main() {
    int resultado = dobrar(5);
    printf("Dobro de 5: %d\n", resultado); // Resultado é 10
    return 0;
}
```

Funções sem Retorno (`void`):

Se a função não precisa retornar nenhum valor, usamos o tipo `void`. Isso é comum em funções que apenas realizam ações, como exibir uma mensagem na tela.

Exemplo de função `void`:

```c
#include <stdio.h>

void saudacao() {
    printf("Olá, bem-vindo!\n");
}

int main() {
    saudacao();
    return 0;
}
```

7. Escopo Local x Escopo Global

O **escopo** de uma variável determina onde ela pode ser acessada no programa.

- Escopo Local: Variáveis declaradas dentro de uma função só podem ser acessadas dentro dessa função. Cada chamada da função cria uma nova versão da variável.
- Escopo Global: Variáveis declaradas fora de qualquer função podem ser acessadas por todo o programa. No entanto, o uso de variáveis globais deve ser controlado para evitar confusão e erros.

```c
#include <stdio.h>

int global = 10; // Variável global

void mostrarGlobal() {
    printf("Variável global: %d\n", global);
}

int main() {
    int local = 5; // Variável local
    printf("Variável local: %d\n", local);
    mostrarGlobal();
    return 0;
}
```

8. Exercícios práticos

- Raiz Quadrada: Crie uma função para calcular a raiz quadrada de um número (utilize a biblioteca `math.h`).
- Cálculo de Potência: Crie uma função para calcular a potência de um número base elevado a um expoente. Utilize a multiplicação em um loop para resolver o problema.
- Conversão de Temperatura: Crie funções que convertam Celsius para Fahrenheit e vice-versa.
- Maior de Dois Números: Crie uma função que receba dois números e retorne o maior deles.

9. Conclusão

Com o uso de funções, conseguimos organizar o código de forma modular, tornando-o mais fácil de ler, depurar e reutilizar. Pratique com os exercícios sugeridos e experimente criar suas próprias funções para diferentes tarefas!