Aula: Funções em C – parte 1

Objetivo da Aula

Ao final desta aula, os alunos deverão ser capazes de:

- Entender conceitos básicos de funções em C.
- Declarar, definir e chamar funções.
- Passar argumentos para funções e retornar valores.
- Diferenciar entre variáveis locais e globais.
- Como usar funções para resolver problemas práticos.

1. Introdução às Funções

Definição: Funções são blocos de código que executam uma tarefa específica dentro de um programa e podem ser reutilizadas em diferentes partes do programa. Elas ajudam a dividir o código em partes menores e mais gerenciáveis, facilitando a leitura e a manutenção.

Podemos entender funções como pequenas "robos" que recebem entradas, processam algo e, muitas vezes, devolvem uma saída. Cada função é independente e pode ser usada em qualquer parte do programa.

Por que usar funções?

- **Reutilização de Código**: Escreva uma vez, use em várias partes do programa.
- **Organização**: Mantém o código organizado e fácil de entender.
- **Manutenção**: Facilita correções e melhorias no código, pois cada função é um bloco separado.
- **Divisão de Tarefas**: Ajuda a dividir um problema complexo em partes menores e mais simples.

Sintaxe Básica de uma Função:

```
tipo_de_retorno nome_da_funcao(tipo1 parametro1, tipo2 parametro2, ...){
    // Corpo da função
    // Código a ser executado
}
```

---

2. Declaração e Definição de Funções

Estrutura Básica de uma Função: Uma função em C tem três partes principais:

1. **Tipo de Retorno**: Define o tipo de dado que a função vai devolver (ex.: `int`, `float`, `void`).
2. **Nome da Função**: Identificador único para a função (ex.: `soma`, `calculaMedia`).
3. **Parâmetros (opcional)**: Dados de entrada para a função, entre parênteses.

Declaração: Informa ao compilador a assinatura da função. Geralmente, é colocada antes da função `main` ou em um arquivo de cabeçalho.

```
int soma(int a, int b); // Declaração da função
```

Definição: Onde a função é realmente implementada com seu corpo.

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

Exemplo Completo:

```
#include <stdio.h>

int soma(int a, int b); // Declaração da função

int main() {
    int resultado = soma(3, 4); // Chamando a função
    printf("Resultado: %d\n", resultado);
    return 0;
}

int soma(int a, int b) { // Definição da função
    return a + b;
}
```

---

3. Chamando Funções

Para chamar uma função, basta usar o nome da função seguido dos parênteses com os argumentos necessários:

```
int resultado = soma(10, 5); // Chamada da função
```

---

4. Parâmetros e Argumentos

Parâmetros: São variáveis definidas na função para receber os dados de entrada. **Argumentos**: São os valores reais passados para a função quando ela é chamada.

Exemplo:

```
#include <stdio.h>

void imprimeMensagem(char mensagem[]); // Declaração da função

int main() {
    imprimeMensagem("Olá, Mundo!"); // Chamando a função com argumento
    return 0;
}

void imprimeMensagem(char mensagem[]) { // Definição com parâmetro
    printf("%s\n", mensagem);
}
```

---

5. Retorno de Valores

Uma função pode retornar um valor usando a palavra-chave `return`. O tipo do valor retornado deve corresponder ao tipo de retorno declarado na função.

Exemplo:

```
#include <stdio.h>

int multiplica(int x, int y); // Declaração da função

int main() {
    int produto = multiplica(6, 7); // Chamada da função
    printf("Produto: %d\n", produto);
    return 0;
}

int multiplica(int x, int y) { // Definição da função
    return x * y; // Retorna o produto de x e y
}
```

Importante: Se uma função não precisa devolver nada, usamos `void` como tipo de retorno.

6. Escopo de Variáveis

- **Variáveis Locais**: Declaradas dentro de uma função e só podem ser usadas nessa função.
- **Variáveis Globais**: Declaradas fora de todas as funções e podem ser acessadas por qualquer função no programa.

Exemplo:

```
#include <stdio.h>

int global = 10; // Variável global

void exemploEscopo() {
    int local = 5; // Variável local
    printf("Global: %d, Local: %d\n", global, local);
}

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

Boas práticas:

- Comece sempre pela declaração da função antes de implementá-la.
- Teste suas funções individualmente para garantir que estão funcionando corretamente.
- Pratique escrevendo funções para diferentes tarefas!

Conclusão

Funções são ferramentas poderosas e fundamentais em C que permitem organizar o código de forma eficiente, facilitando a escrita, leitura, e manutenção dos programas. Elas permitem a reutilização de código,

dividem o programa em partes menores e ajudam na solução de problemas complexos por meio de abordagens estruturadas.

7. Exemplos Práticos e Exercícios com Respostas

Exemplos básicos

Exemplo 1: Função para Calcular o Fatorial: O fatorial de um número `n` é o produto de todos os inteiros positivos até `n`. Por exemplo, o fatorial de 4 é `4 * 3 * 2 * 1 = 24`.

```
#include <stdio.h>

int fatorial(int n);

int main() {
    int numero;
    printf("Digite um número para calcular o fatorial: ");
    scanf("%d", &numero);
    printf("Fatorial de %d é %d\n", numero, fatorial(numero));
    return 0;
}

int fatorial(int n) {
    if (n == 0) {
        return 1; // O fatorial de 0 é 1
    } else {
        return n * fatorial(n - 1); // Chamada recursiva
    }
}
```

Exemplo 2: Função para Verificar Par ou Ímpar: Uma função que verifica se um número é par ou ímpar.

```
#include <stdio.h>

void verificaParImpar(int num);

int main() {
    int numero;
    printf("Digite um número: ");
    scanf("%d", &numero);
    verificaParImpar(numero);
    return 0;
}

void verificaParImpar(int num) {
    if (num % 2 == 0) {
        printf("%d é par.\n", num);
    } else {
        printf("%d é ímpar.\n", num);
    }
}
```

Exercícios com (uso de Matrizes)

Exercício 3 Pontuação de Jogos com Função: Criar uma função para calcular a soma das pontuações dos jogadores.

```
#include <stdio.h>

#define JOGADORES 3
#define JOGOS 5

void lerPontos(int pontos[JOGADORES][JOGOS]);
void calcularSoma(int pontos[JOGADORES][JOGOS], int total[JOGADORES]);

int main() {
    int pontos[JOGADORES][JOGOS];
    int total[JOGADORES] = {0, 0, 0};

    lerPontos(pontos);
    calcularSoma(pontos, total);

    for (int i = 0; i < JOGADORES; i++) {
        printf("Total de pontos do jogador %d: %d\n", i + 1, total[i]);
    }

    return 0;
}

void lerPontos(int pontos[JOGADORES][JOGOS]) {
    for (int i = 0; i < JOGADORES; i++) {
        printf("Insira as pontuações do jogador %d:\n", i + 1);
        for (int j = 0; j < JOGOS; j++) {
            scanf("%d", &pontos[i][j]);
        }
    }
}

void calcularSoma(int pontos[JOGADORES][JOGOS], int total[JOGADORES]) {
    for (int i = 0; i < JOGADORES; i++) {
        for (int j = 0; j < JOGOS; j++) {
            total[i] += pontos[i][j];
        }
    }
}
```

Exercício 4: Ganhador do Jogo com Função: Criar uma função para determinar o jogador com a maior pontuação total.

```
#include <stdio.h>

#define JOGADORES 4
#define PARTIDAS 3

void lerPontos(int pontos[JOGADORES][PARTIDAS]);
int encontrarGanhador(int pontos[JOGADORES][PARTIDAS]);

int main() {
    int pontos[JOGADORES][PARTIDAS];

    lerPontos(pontos);
    int ganhador = encontrarGanhador(pontos);

    printf("O jogador %d é o ganhador.\n", ganhador + 1);

    return 0;
}

void lerPontos(int pontos[JOGADORES][PARTIDAS]) {
    for (int i = 0; i < JOGADORES; i++) {
        printf("Insira as pontuações do jogador %d:\n", i + 1);
        for (int j = 0; j < PARTIDAS; j++) {
            scanf("%d", &pontos[i][j]);
        }
    }
}

int encontrarGanhador(int pontos[JOGADORES][PARTIDAS]) {
    int total[JOGADORES] = {0};
    int maior = 0, ganhador = 0;

    for (int i = 0; i < JOGADORES; i++) {
        for (int j = 0; j < PARTIDAS; j++) {
            total[i] += pontos[i][j];
        }
        if (total[i] > maior) {
            maior = total[i];
            ganhador = i;
        }
    }
    return ganhador;
}
```

Exercícios com Tabelas de Preços

Exercício 5: Total de Preços de Produtos com Função: Criar uma função para calcular o total dos preços de produtos em diferentes lojas.

```
#include <stdio.h>

#define PRODUTOS 3
#define LOJAS 4

void lerPrecos(float precos[PRODUTOS][LOJAS]);
void calcularTotal(float precos[PRODUTOS][LOJAS], float
total[PRODUTOS]);

int main() {
    float precos[PRODUTOS][LOJAS];
    float total[PRODUTOS] = {0, 0, 0};

    lerPrecos(precos);
    calcularTotal(precos, total);

    for (int i = 0; i < PRODUTOS; i++) {
        printf("Total de preços do produto %d: %.2f\n", i + 1,
total[i]);
    }

    return 0;
}

void lerPrecos(float precos[PRODUTOS][LOJAS]) {
    for (int i = 0; i < PRODUTOS; i++) {
        printf("Insira os preços do produto %d nas 4 lojas:\n", i + 1);
        for (int j = 0; j < LOJAS; j++) {
            scanf("%f", &precos[i][j]);
        }
    }
}

void calcularTotal(float precos[PRODUTOS][LOJAS], float total[PRODUTOS])
{
    for (int i = 0; i < PRODUTOS; i++) {
        for (int j = 0; j < LOJAS; j++) {
            total[i] += precos[i][j];
        }
    }
}
```

Exercício 6: Produto Mais Barato com Função: Criar uma função para encontrar o produto com o menor preço total entre todas as lojas.

```
#include <stdio.h>

#define PRODUTOS 5
#define LOJAS 3

void lerPrecos(float precos[PRODUTOS][LOJAS]);
int encontrarMaisBarato(float precos[PRODUTOS][LOJAS]);

int main() {
    float precos[PRODUTOS][LOJAS];

    lerPrecos(precos);
    int maisBarato = encontrarMaisBarato(precos);

    printf("O produto %d é o mais barato.\n", maisBarato + 1);

    return 0;
}

void lerPrecos(float precos[PRODUTOS][LOJAS]) {
    for (int i = 0; i < PRODUTOS; i++) {
        printf("Insira os preços do produto %d nas 3 lojas:\n", i + 1);
        for (int j = 0; j < LOJAS; j++) {
            scanf("%f", &precos[i][j]);
        }
    }
}

int encontrarMaisBarato(float precos[PRODUTOS][LOJAS]) {
    float total[PRODUTOS] = {0, 0, 0, 0, 0};
    int maisBarato = 0;

    for (int i = 0; i < PRODUTOS; i++) {
        for (int j = 0; j < LOJAS; j++) {
            total[i] += precos[i][j];
        }
        if (total[i] < total[maisBarato]) {
            maisBarato = i;
        }
    }

    return maisBarato;
}
```

Exercícios Diversos com Vetores e Matrizes

Exercício 7: Soma de Matrizes com Função: Criar uma função para somar duas matrizes e retornar a matriz resultante.

```
#include <stdio.h>

#define TAM 2

void lerMatriz(int matriz[TAM][TAM]);
void somaMatrizes(int matriz1[TAM][TAM], int matriz2[TAM][TAM], int
soma[TAM][TAM]);
void imprimirMatriz(int matriz[TAM][TAM]);

int main() {
    int matriz1[TAM][TAM], matriz2[TAM][TAM], soma[TAM][TAM];

    printf("Insira os elementos da primeira matriz 2x2:\n");
    lerMatriz(matriz1);
    printf("Insira os elementos da segunda matriz 2x2:\n");
    lerMatriz(matriz2);

    somaMatrizes(matriz1, matriz2, soma);

    printf("Matriz resultante da soma:\n");
    imprimirMatriz(soma);

    return 0;
}

void lerMatriz(int matriz[TAM][TAM]) {
    for (int i = 0; i < TAM; i++) {
        for (int j = 0; j < TAM; j++) {
            scanf("%d", &matriz[i][j]);
        }
    }
}

void somaMatrizes(int matriz1[TAM][TAM], int matriz2[TAM][TAM], int
soma[TAM][TAM]) {
    for (int i = 0; i < TAM; i++) {
        for (int j = 0; j < TAM; j++) {
            soma[i][j] = matriz1[i][j] + matriz2[i][j];
        }
    }
}

void imprimirMatriz(int matriz[TAM][TAM]) {
    for (int i = 0; i < TAM; i++) {
        for (int j = 0; j < TAM; j++) {
            printf("%d ", matriz[i][j]);
        }
        printf("\n");
    }
}
```

Exercício 8: Diagonal Principal de uma Matriz com Função: Criar uma função para exibir os elementos da diagonal principal de uma matriz.

```
#include <stdio.h>

#define TAM 4

void lerMatriz(int matriz[TAM][TAM]);
void exibirDiagonalPrincipal(int matriz[TAM][TAM]);

int main() {
    int matriz[TAM][TAM];

    printf("Insira os elementos da matriz 4x4:\n");
    lerMatriz(matriz);

    printf("Diagonal principal:\n");
    exibirDiagonalPrincipal(matriz);

    return 0;
}

void lerMatriz(int matriz[TAM][TAM]) {
    for (int i = 0; i < TAM; i++) {
        for (int j = 0; j < TAM; j++) {
            scanf("%d", &matriz[i][j]);
        }
    }
}

void exibirDiagonalPrincipal(int matriz[TAM][TAM]) {
    for (int i = 0; i < TAM; i++) {
        printf("%d ", matriz[i][i]);
    }
    printf("\n");
}
```

Exercício 9: Função para Verificar Matriz Identidade: Criar uma função para verificar se uma matriz é uma matriz identidade.

```
#include <stdio.h>

#define TAM 4

void lerMatriz(int matriz[TAM][TAM]);
int ehIdentidade(int matriz[TAM][TAM]);

int main() {
    int matriz[TAM][TAM];

    printf("Insira os elementos da matriz 4x4:\n");
    lerMatriz(matriz);

    if (ehIdentidade(matriz)) {
        printf("A matriz é uma identidade.\n");
    } else {
        printf("A matriz não é uma identidade.\n");
    }

    return 0;
}

void lerMatriz(int matriz[TAM][TAM]) {
    for (int i = 0; i < TAM; i++) {
        for (int j = 0; j < TAM; j++) {
            scanf("%d", &matriz[i][j]);
        }
    }
}

int ehIdentidade(int matriz[TAM][TAM]) {
    for (int i = 0; i < TAM; i++) {
        for (int j = 0; j < TAM; j++) {
            if ((i == j && matriz[i][j] != 1) || (i != j && matriz[i][j]
!= 0)) {
                return 0;
            }
        }
    }
    return 1;
}
```