Material de Apoio - Semana 5

Funções Parte 1

Introdução a Técnicas de Programação (2025.2)

Objetivos de Aprendizagem

Ao final desta semana, você será capaz de:

- Compreender o conceito e a importância das funções em C
- Criar e utilizar funções com diferentes tipos de parâmetros e retornos
- Distinguir entre variáveis locais e globais (escopo)
- Aplicar boas práticas no desenvolvimento de funções
- Passar vetores como parâmetros para funções
- Utilizar argumentos de linha de comando na função main

---

1. Introdução às Funções

1.1 Por que Precisamos de Funções?

Imagine que você precise calcular a potência de vários números diferentes em seu programa. Sem funções, você teria que repetir o mesmo código várias vezes:

```c
#include <stdio.h>

int main() {
    // Calcular 2^3
    int resultado1 = 1;
    for (int i = 0; i < 3; i++) {
        resultado1 *= 2;
    }
    printf("2^3 = %d\n", resultado1);

    // Calcular 4^2
    int resultado2 = 1;
    for (int i = 0; i < 2; i++) {
        resultado2 *= 4;
    }
    printf("4^2 = %d\n", resultado2);

    // Mais código repetitivo...

    return 0;
}
```

Problemas desta abordagem:

- Código repetitivo e extenso
- Dificulta manutenção
- Propenso a erros
- Falta de reutilização
- Código menos organizado

1.2 O Conceito de Função

Uma **função** é um bloco de código que:

- Realiza uma tarefa específica
- Pode receber dados de entrada (parâmetros)
- Pode retornar um resultado
- Pode ser chamada quantas vezes necessário
- Facilita a organização e reutilização do código

1.3 Funções que Já Utilizamos

Você já utilizou várias funções da linguagem C:

```c
printf("Olá mundo!");        // Escreve na saída padrão
scanf("%d", &numero);        // Lê da entrada padrão
sqrt(16);                    // Calcula raiz quadrada
pow(2, 3);                   // Calcula potência
strlen("texto");             // Calcula comprimento de string
```

Todas essas funções estão organizadas em **bibliotecas** que incluímos com `#include`.

---

2. Sintaxe e Estrutura das Funções

2.1 Anatomia de uma Função

```c
tipo_retorno nome_funcao(tipo_param1 param1, tipo_param2 param2) {
    // declarações de variáveis locais

    // instruções da função

    return valor_retorno;  // opcional, depende do tipo de retorno
}
```

Componentes:

- **Tipo de retorno**: Tipo do valor que a função retorna (`int`, `float`, `void`, etc.)
- **Nome da função**: Identificador único da função
- **Parâmetros**: Dados que a função recebe (opcionais)

- **Corpo da função**: Código que será executado
- **return**: Especifica o valor de retorno e encerra a função

2.2 Assinatura da Função

A **assinatura** de uma função é composta por:

- Tipo de retorno
- Nome da função
- Tipos e ordem dos parâmetros

Exemplo: `int soma(int a, int b)`

2.3 Declaração vs Definição

Declaração (protótipo): Informa ao compilador sobre a existência da função

```c
int soma(int a, int b);  // Apenas declara
```

Definição: Implementa o código da função

```c
int soma(int a, int b) {  // Define o comportamento
    return a + b;
}
```

---

3. Exemplos Básicos de Funções

3.1 Função Simples de Soma

```c
#include <stdio.h>

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

int main() {
    // Chamadas da função
    printf("3 + 5 = %d\n", soma(3, 5));
    printf("10 + 15 = %d\n", soma(10, 15));

    // Armazenando o resultado
    int resultado = soma(7, 8);
    printf("7 + 8 = %d\n", resultado);

    return 0;
}
```

3.2 Função que Não Retorna Valor (`void`)

```c
#include <stdio.h>

void exibirMensagem(char mensagem[]) {
    printf("=== %s ===\n", mensagem);
}

void exibirTabuada(int numero) {
    printf("Tabuada do %d:\n", numero);
    for (int i = 1; i <= 10; i++) {
        printf("%d x %d = %d\n", numero, i, numero * i);
    }
}

int main() {
    exibirMensagem("CALCULADORA");
    exibirTabuada(5);
    exibirMensagem("FIM");

    return 0;
}
```

3.3 Função com Lógica Mais Complexa

```c
#include <stdio.h>

// Função que verifica se um número é primo
int ehPrimo(int numero) {
    if (numero < 2) {
        return 0;  // Não é primo
    }

    for (int i = 2; i < numero; i++) {
        if (numero % i == 0) {
            return 0;  // Não é primo
        }
    }

    return 1;  // É primo
}

int main() {
    int num;

    printf("Digite um número: ");
    scanf("%d", &num);
```

```
    if (ehPrimo(num)) {
        printf("%d é primo.\n", num);
    } else {
        printf("%d não é primo.\n", num);
    }

    return 0;
}
```

---

4. Escopo de Variáveis

4.1 Variáveis Locais

Variáveis locais são declaradas dentro de funções e só existem durante a execução da função:

```
#include <stdio.h>

void funcao1() {
    int x = 10;  // Variável local de funcao1
    printf("Em funcao1: x = %d\n", x);
}

void funcao2() {
    int x = 20;  // Variável local diferente de funcao2
    printf("Em funcao2: x = %d\n", x);
}

int main() {
    int x = 5;   // Variável local de main

    printf("Em main: x = %d\n", x);
    funcao1();
    funcao2();
    printf("Em main novamente: x = %d\n", x);

    return 0;
}
```

Saída:

```
Em main: x = 5
Em funcao1: x = 10
Em funcao2: x = 20
Em main novamente: x = 5
```

4.2 Variáveis Globais

Variáveis globais são declaradas fora de todas as funções e podem ser acessadas por qualquer função:

```c
#include <stdio.h>

int contador = 0;  // Variável global

void incrementar() {
    contador++;
    printf("Contador incrementado para: %d\n", contador);
}

void decrementar() {
    contador--;
    printf("Contador decrementado para: %d\n", contador);
}

int main() {
    printf("Contador inicial: %d\n", contador);

    incrementar();
    incrementar();
    decrementar();

    printf("Contador final: %d\n", contador);
    return 0;
}
```

4.3 Boas Práticas de Escopo

✅ **Prefira variáveis locais:**

- Mais seguro
- Evita efeitos colaterais
- Facilita depuração
- Melhora legibilidade

⚠ **Use variáveis globais com moderação:**

- Apenas quando realmente necessário
- Para constantes ou configurações
- Para dados compartilhados entre muitas funções

---

5. Passagem de Parâmetros

5.1 Passagem por Valor

Em C, os parâmetros são passados **por valor**, ou seja, uma cópia do valor é criada:

```c
#include <stdio.h>

void tentarModificar(int x) {
    x = x * 2;
    printf("Dentro da função: x = %d\n", x);
}

int main() {
    int numero = 10;

    printf("Antes da chamada: numero = %d\n", numero);
    tentarModificar(numero);
    printf("Depois da chamada: numero = %d\n", numero);

    return 0;
}
```

Saída:

```
Antes da chamada: numero = 10
Dentro da função: x = 20
Depois da chamada: numero = 10
```

5.2 Exemplo Prático: Calculadora

```c
#include <stdio.h>

float somar(float a, float b) {
    return a + b;
}

float subtrair(float a, float b) {
    return a - b;
}

float multiplicar(float a, float b) {
    return a * b;
}

float dividir(float a, float b) {
    if (b != 0) {
        return a / b;
    } else {
        printf("Erro: Divisão por zero!\n");
        return 0;
    }
}
```

```c
void exibirMenu() {
    printf("\n=== CALCULADORA ===\n");
    printf("1. Somar\n");
    printf("2. Subtrair\n");
    printf("3. Multiplicar\n");
    printf("4. Dividir\n");
    printf("5. Sair\n");
    printf("Escolha uma opção: ");
}

int main() {
    int opcao;
    float num1, num2, resultado;

    do {
        exibirMenu();
        scanf("%d", &opcao);

        if (opcao >= 1 && opcao <= 4) {
            printf("Digite dois números: ");
            scanf("%f %f", &num1, &num2);

            switch (opcao) {
                case 1:
                    resultado = somar(num1, num2);
                    printf("Resultado: %.2f\n", resultado);
                    break;
                case 2:
                    resultado = subtrair(num1, num2);
                    printf("Resultado: %.2f\n", resultado);
                    break;
                case 3:
                    resultado = multiplicar(num1, num2);
                    printf("Resultado: %.2f\n", resultado);
                    break;
                case 4:
                    resultado = dividir(num1, num2);
                    if (num2 != 0) {
                        printf("Resultado: %.2f\n", resultado);
                    }
                    break;
            }
        } else if (opcao != 5) {
            printf("Opção inválida!\n");
        }

    } while (opcao != 5);

    printf("Calculadora encerrada.\n");
    return 0;
}
```

6. Funções com Vetores

6.1 Passando Vetores como Parâmetros

Vetores são passados de forma diferente - por referência (endereço):

```c
#include <stdio.h>

// Opção 1: Usando notação de vetor
void exibirVetor1(int vetor[], int tamanho) {
    printf("Vetor: ");
    for (int i = 0; i < tamanho; i++) {
        printf("%d ", vetor[i]);
    }
    printf("\n");
}

// Opção 2: Usando notação de ponteiro (equivalente)
void exibirVetor2(int *vetor, int tamanho) {
    printf("Vetor: ");
    for (int i = 0; i < tamanho; i++) {
        printf("%d ", vetor[i]);
    }
    printf("\n");
}

int main() {
    int numeros[] = {3, 7, 1, 9, 5};
    int tamanho = 5;

    exibirVetor1(numeros, tamanho);
    exibirVetor2(numeros, tamanho);

    return 0;
}
```

6.2 Modificando Vetores em Funções

Como vetores são passados por referência, podemos modificá-los:

```c
#include <stdio.h>

void dobrarElementos(int vetor[], int tamanho) {
    for (int i = 0; i < tamanho; i++) {
        vetor[i] = vetor[i] * 2;
    }
}

void preencherVetor(int vetor[], int tamanho, int valor) {
    for (int i = 0; i < tamanho; i++) {
```

```
        vetor[i] = valor;
    }
}

int somarElementos(int vetor[], int tamanho) {
    int soma = 0;
    for (int i = 0; i < tamanho; i++) {
        soma += vetor[i];
    }
    return soma;
}

int main() {
    int numeros[5];

    // Preencher o vetor
    preencherVetor(numeros, 5, 3);
    printf("Após preencher: ");
    for (int i = 0; i < 5; i++) {
        printf("%d ", numeros[i]);
    }
    printf("\n");

    // Dobrar os elementos
    dobrarElementos(numeros, 5);
    printf("Após dobrar: ");
    for (int i = 0; i < 5; i++) {
        printf("%d ", numeros[i]);
    }
    printf("\n");

    // Somar os elementos
    int soma = somarElementos(numeros, 5);
    printf("Soma dos elementos: %d\n", soma);

    return 0;
}
```

6.3 Exemplo Avançado: Busca e Ordenação

```
#include <stdio.h>

int buscarElemento(int vetor[], int tamanho, int elemento) {
    for (int i = 0; i < tamanho; i++) {
        if (vetor[i] == elemento) {
            return i;  // Retorna o índice onde foi encontrado
        }
    }
    return -1;  // Não encontrado
}
```

```c
void ordenarVetor(int vetor[], int tamanho) {
    // Algoritmo Bubble Sort simples
    for (int i = 0; i < tamanho - 1; i++) {
        for (int j = 0; j < tamanho - 1 - i; j++) {
            if (vetor[j] > vetor[j + 1]) {
                // Trocar elementos
                int temp = vetor[j];
                vetor[j] = vetor[j + 1];
                vetor[j + 1] = temp;
            }
        }
    }
}

void exibirVetor(int vetor[], int tamanho) {
    for (int i = 0; i < tamanho; i++) {
        printf("%d ", vetor[i]);
    }
    printf("\n");
}

int main() {
    int numeros[] = {64, 34, 25, 12, 22, 11, 90};
    int tamanho = 7;
    int elemento_busca = 25;

    printf("Vetor original: ");
    exibirVetor(numeros, tamanho);

    // Buscar elemento
    int posicao = buscarElemento(numeros, tamanho, elemento_busca);
    if (posicao != -1) {
        printf("Elemento %d encontrado na posição %d\n", elemento_busca,
posicao);
    } else {
        printf("Elemento %d não encontrado\n", elemento_busca);
    }

    // Ordenar vetor
    ordenarVetor(numeros, tamanho);
    printf("Vetor ordenado: ");
    exibirVetor(numeros, tamanho);

    return 0;
}
```

---

7. A Função main e Argumentos de Linha de Comando

7.1 Variações da Função main

```c
// Versão simples
int main() {
    return 0;
}

// Versão com argumentos de linha de comando
int main(int argc, char *argv[]) {
    return 0;
}

// Versão alternativa (menos comum)
int main(int argc, char **argv) {
    return 0;
}
```

7.2 Argumentos de Linha de Comando

- **argc**: Quantidade de argumentos (incluindo o nome do programa)
- **argv**: Vetor de strings com os argumentos

```c
#include <stdio.h>

int main(int argc, char *argv[]) {
    printf("Número de argumentos: %d\n", argc);

    for (int i = 0; i < argc; i++) {
        printf("Argumento %d: %s\n", i, argv[i]);
    }

    return 0;
}
```

Exemplo de execução:

```
$ ./programa hello world 123
Número de argumentos: 4
Argumento 0: ./programa
Argumento 1: hello
Argumento 2: world
Argumento 3: 123
```

7.3 Exemplo Prático: Calculadora por Linha de Comando

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

```c
int main(int argc, char *argv[]) {
    if (argc != 4) {
        printf("Uso: %s <num1> <operação> <num2>\n", argv[0]);
        printf("Operações: +, -, *, /\n");
        return 1;
    }

    double num1 = atof(argv[1]);
    char operacao = argv[2][0];
    double num2 = atof(argv[3]);
    double resultado;

    switch (operacao) {
        case '+':
            resultado = num1 + num2;
            break;
        case '-':
            resultado = num1 - num2;
            break;
        case '*':
            resultado = num1 * num2;
            break;
        case '/':
            if (num2 != 0) {
                resultado = num1 / num2;
            } else {
                printf("Erro: Divisão por zero!\n");
                return 1;
            }
            break;
        default:
            printf("Operação inválida: %c\n", operacao);
            return 1;
    }

    printf("%.2f %c %.2f = %.2f\n", num1, operacao, num2, resultado);
    return 0;
}
```

---

8. Boas Práticas com Funções

8.1 Nomenclatura de Funções

✅ **Bons nomes:**

```c
int calcularIdade(int anoNascimento);
void exibirMenu();
float converterCelsiusParaFahrenheit(float celsius);
int ehNumeroPar(int numero);
```

❌ **Nomes ruins:**

```c
int calc(int x);
void f1();
float conv(float t);
int check(int n);
```

8.2 Tamanho e Responsabilidade

✅ **Funções focadas:**

```c
// Cada função tem uma responsabilidade específica
int lerNumero() {
    int num;
    scanf("%d", &num);
    return num;
}

void exibirResultado(int resultado) {
    printf("Resultado: %d\n", resultado);
}

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

❌ **Função fazendo demais:**

```c
// Função que faz tudo - difícil de entender e manter
void programa() {
    int num;
    printf("Digite um número: ");
    scanf("%d", &num);

    int fatorial = 1;
    for (int i = 1; i <= num; i++) {
        fatorial *= i;
    }

    printf("Resultado: %d\n", fatorial);

    // ... mais código não relacionado
}
```

8.3 Documentação de Funções

```c
/**
 * Calcula o fatorial de um número inteiro não negativo
 * @param n: número inteiro não negativo
 * @return: fatorial de n, ou -1 se n for negativo
 */
int calcularFatorial(int n) {
    if (n < 0) {
        return -1;  // Erro: número negativo
    }

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

/**
 * Verifica se um número é primo
 * @param numero: número inteiro a ser verificado
 * @return: 1 se for primo, 0 se não for primo
 */
int ehPrimo(int numero) {
    if (numero < 2) return 0;

    for (int i = 2; i < numero; i++) {
        if (numero % i == 0) {
            return 0;
        }
    }
    return 1;
}
```

---

9. Problemas Clássicos com Funções

9.1 Sistema de Notas de Alunos

```c
#include <stdio.h>

float calcularMedia(float n1, float n2, float n3) {
    return (n1 + n2 + n3) / 3.0;
}

char obterConceito(float media) {
    if (media >= 9.0) return 'A';
```

```
    else if (media >= 7.0) return 'B';
    else if (media >= 5.0) return 'C';
    else return 'D';
}

void exibirSituacao(char conceito) {
    printf("Conceito: %c - ", conceito);
    switch (conceito) {
        case 'A': printf("Excelente\n"); break;
        case 'B': printf("Bom\n"); break;
        case 'C': printf("Regular\n"); break;
        case 'D': printf("Insuficiente\n"); break;
    }
}

int main() {
    float nota1, nota2, nota3;

    printf("Digite as três notas: ");
    scanf("%f %f %f", &nota1, &nota2, &nota3);

    float media = calcularMedia(nota1, nota2, nota3);
    char conceito = obterConceito(media);

    printf("Média: %.2f\n", media);
    exibirSituacao(conceito);

    return 0;
}
```

9.2 Conversor de Temperaturas

```
#include <stdio.h>

float celsiusParaFahrenheit(float celsius) {
    return (celsius * 9.0 / 5.0) + 32.0;
}

float fahrenheitParaCelsius(float fahrenheit) {
    return (fahrenheit - 32.0) * 5.0 / 9.0;
}

float celsiusParaKelvin(float celsius) {
    return celsius + 273.15;
}

float kelvinParaCelsius(float kelvin) {
    return kelvin - 273.15;
}

void exibirMenu() {
```

```c
    printf("\n=== CONVERSOR DE TEMPERATURAS ===\n");
    printf("1. Celsius para Fahrenheit\n");
    printf("2. Fahrenheit para Celsius\n");
    printf("3. Celsius para Kelvin\n");
    printf("4. Kelvin para Celsius\n");
    printf("5. Sair\n");
    printf("Escolha uma opção: ");
}

int main() {
    int opcao;
    float temperatura, resultado;

    do {
        exibirMenu();
        scanf("%d", &opcao);

        if (opcao >= 1 && opcao <= 4) {
            printf("Digite a temperatura: ");
            scanf("%f", &temperatura);

            switch (opcao) {
                case 1:
                    resultado = celsiusParaFahrenheit(temperatura);
                    printf("%.2f°C = %.2f°F\n", temperatura, resultado);
                    break;
                case 2:
                    resultado = fahrenheitParaCelsius(temperatura);
                    printf("%.2f°F = %.2f°C\n", temperatura, resultado);
                    break;
                case 3:
                    resultado = celsiusParaKelvin(temperatura);
                    printf("%.2f°C = %.2fK\n", temperatura, resultado);
                    break;
                case 4:
                    resultado = kelvinParaCelsius(temperatura);
                    printf("%.2fK = %.2f°C\n", temperatura, resultado);
                    break;
            }
        } else if (opcao != 5) {
            printf("Opção inválida!\n");
        }

    } while (opcao != 5);

    return 0;
}
```

---

10. Exercícios Resolvidos

10.1 Maior de Dois Números

```c
#include <stdio.h>

int maiorDeDois(int a, int b) {
    if (a > b) {
        return a;
    } else {
        return b;
    }
    // Versão mais concisa: return (a > b) ? a : b;
}

int main() {
    printf("Maior entre 5 e 2: %d\n", maiorDeDois(5, 2));
    printf("Maior entre 3 e 8: %d\n", maiorDeDois(3, 8));

    return 0;
}
```

10.2 Escrever Intervalo

```c
#include <stdio.h>

void escreverIntervalo(int a, int b) {
    if (a <= b) {
        for (int i = a; i <= b; i++) {
            printf("%d ", i);
        }
    } else {
        for (int i = a; i >= b; i--) {
            printf("%d ", i);
        }
    }
    printf("\n");
}

int main() {
    escreverIntervalo(2, 5);   // 2 3 4 5
    escreverIntervalo(8, 3);   // 8 7 6 5 4 3

    return 0;
}
```

10.3 Somatório

```c
#include <stdio.h>

int somatorio(int a, int b) {
    int soma = 0;
```

```
    for (int i = a; i <= b; i++) {
        soma += i;
    }
    return soma;
}

int main() {
    printf("Somatório de 1 a 5: %d\n", somatorio(1, 5));     // 15
    printf("Somatório de 10 a 40: %d\n", somatorio(10, 40)); // 775

    return 0;
}
```

---

11. Dicas para a Lista de Exercícios

11.1 Estratégias de Desenvolvimento

1. **Planeje antes de programar:**
    - Defina claramente o que a função deve fazer
    - Identifique os parâmetros necessários
    - Determine o tipo de retorno
2. **Comece simples:**
    - Implemente a versão básica primeiro
    - Teste com casos simples
    - Adicione melhorias gradualmente
3. **Use nomes descritivos:**
    - Funções: verbos que descrevem a ação
    - Parâmetros: substantivos que representam os dados

11.2 Checklist de Verificação

- [ ] A função tem uma responsabilidade bem definida?
- [ ] O nome da função é claro e descritivo?
- [ ] Os parâmetros são necessários e suficientes?
- [ ] O tipo de retorno está correto?
- [ ] A função trata casos especiais adequadamente?
- [ ] O código está bem comentado?
- [ ] A função foi testada com diferentes entradas?

11.3 Padrões Comuns de Funções

Funções de cálculo:

```
int calcularFatorial(int n);
float calcularMedia(float valores[], int tamanho);
```

Funções de validação:

```
int ehPrimo(int numero);
int ehPar(int numero);
int ehPositivo(float numero);
```

Funções de entrada/saída:

```
int lerNumeroInteiro();
void exibirResultado(float resultado);
void exibirMenu();
```

Funções de manipulação:

```
void ordenarVetor(int vetor[], int tamanho);
int buscarElemento(int vetor[], int tamanho, int elemento);
```

---

12. Preparação para a Próxima Semana

Na **Semana 6**, estudaremos:

- Conceitos fundamentais de vetores (arrays)
- Declaração e inicialização de vetores
- Acesso e modificação de elementos
- Vetores como contadores e marcadores
- Detalhes implementacionais (VLA, segmentation fault)

Exercício preparatório: Pense em como você poderia usar as funções aprendidas nesta semana para trabalhar com conjuntos de dados. Como uma função poderia processar um vetor de números para encontrar o maior valor? Como organizar um programa que trabalha com múltiplos vetores usando funções especializadas?

---

13. Recursos Adicionais

13.1 Bibliotecas Úteis para Funções

```
#include <stdio.h>   // printf, scanf
#include <stdlib.h>  // atoi, atof, malloc, free
```

```c
#include <string.h>  // strlen, strcmp, strcpy
#include <math.h>    // sqrt, pow, sin, cos
#include <time.h>    // time, clock
```

13.2 Ferramentas para Depuração de Funções

```
# Compilar com símbolos de depuração
gcc -g -Wall -o programa programa.c

# Usar GDB para depurar
gdb ./programa

# Comandos úteis no GDB
(gdb) break nome_funcao
(gdb) step
(gdb) print variavel
(gdb) backtrace
```

13.3 Exercícios Extras para Prática

1. Crie uma função que calcule o MDC (Máximo Divisor Comum) de dois números
2. Implemente uma função que converta um número decimal para binário
3. Desenvolva uma função que verifique se uma palavra é palíndromo
4. Crie um conjunto de funções para operações com números complexos
5. Implemente uma função que calcule a sequência de Fibonacci até o n-ésimo termo

Lembre-se: Funções bem projetadas são a base de programas organizados e fáceis de manter. Pratique criando funções pequenas, focadas e reutilizáveis!