# Programação de Computadores

Raoni F. S. Teixeira

Aulas 10 e 11 - Funções

## 1 Introdução

Considere o seguinte problema: um número composto $n$ é **produto de dois números primos** se existirem dois números primos $p$ e $q$ tais que $n = p \times q$. Queremos escrever um programa que leia um número $n$ e decida se ele é produto de dois números primos.

Uma primeira tentativa de solução seria escrever um código longo que verifica se cada possível fator é primo:

```
int n, p, q, d, p_eh_primo, q_eh_primo, eh_produto = 0;
scanf("%d", &n);
for (p = 1; p < n; p++) {
    if (n % p == 0) {
        q = n / p;
        p_eh_primo = 1;
        for (d = 2; d < p; d++) {
            if (p % d == 0)
                p_eh_primo = 0;
        }
        q_eh_primo = 1;
        for (d = 2; d < q; d++) {
            if (q % d == 0)
                q_eh_primo = 0;
        }
        if (p_eh_primo && q_eh_primo)
            eh_produto = 1;
    }
}
```

Este código apresenta vários problemas:

- Escrevemos dois blocos de código muito parecidos (para verificar se $p$ e $q$ são primos)
- O código se torna muito grande e difícil de ler

- É difícil ver o que cada bloco faz
- Declaramos muitas variáveis auxiliares
- Se houver um erro na lógica de verificação de primalidade, precisamos corrigi-lo em dois lugares

E se pudéssemos simplesmente "perguntar" se um número é primo? Com funções, podemos fazer exatamente isso:

```
int n, p, q, eh_produto = 0;
scanf("%d", &n);
for (p = 1; p < n; p++) {
    if (n % p == 0) {
        q = n / p;
        if (eh_primo(p) && eh_primo(q)) {
            eh_produto = 1;
        }
    }
}
```

O código ficou muito mais claro e conciso! Mas o compilador C não entende automaticamente o que significa `eh_primo`. Precisamos *ensinar* ao compilador como responder a essa pergunta, definindo uma função.

# 2 Funções

Uma **função** é uma estrutura que agrupa um conjunto de comandos que são executados quando a função é chamada. Funções são fundamentais na programação porque permitem:

1. Evitar que os blocos do programa fiquem grandes demais e difíceis de ler e entender.
2. Separar o programa em partes que possam ser logicamente compreendidas de forma isolada.
3. Permitir o reaproveitamento de código já construído (por você ou por outros programadores).
4. Evitar que um trecho de código seja repetido várias vezes dentro de um mesmo programa, minimizando erros e facilitando alterações.

As funções podem retornar um valor ao final de sua execução. Por exemplo, a função `sqrt(4)` da biblioteca matemática retorna o valor 2.0.

## 2.1 Declarando uma Função

Uma função é declarada da seguinte forma:

```
tipo nome(tipo parametro1, ..., tipo parametroN) {
    comandos;
    return valor_de_retorno;
}
```

Onde:

- **tipo**: Determina qual será o tipo do valor de retorno da função.
- **nome**: Identificador da função, usado para chamá-la.
- **parâmetros**: Variáveis que serão utilizadas pela função. Essas variáveis são inicializadas com valores na chamada da função.
- **return**: Comando que encerra a execução da função e retorna um valor.

Voltando ao nosso exemplo, podemos ensinar ao compilador como verificar se um número é primo:

```
int eh_primo(int p) {
    int d, primo;
    primo = 1;
    for (d = 2; d < p; d++) {
        if (p % d == 0)
            primo = 0;
    }
    return primo;
}
```

Esta função recebe um inteiro `p` como parâmetro e retorna 1 (verdadeiro) se `p` for primo, ou 0 (falso) caso contrário.

## 2.2 Exemplo Simples de Função

A função abaixo soma dois valores passados como parâmetros:

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

Note que:

- O valor de retorno é do mesmo tipo definido na declaração da função (`int`).
- Quando o comando `return` é executado, a função para de executar imediatamente e retorna o valor indicado.
- Nada após o `return` será executado.

## 2.3 Invocando uma Função

Para usar (ou *invocar*) uma função, basta chamá-la pelo nome e fornecer os valores para seus parâmetros:

```
int x = soma(4, 2);  // x recebe 6
```

O resultado de uma chamada de função é uma expressão e pode ser usado em qualquer lugar que aceite uma expressão:

```
printf("Media: %d\n", soma(a, b) / 2);
printf("Soma de 5 e 10: %d\n", soma(5, 10));
```

## 2.4 Programa Completo com Função

```
#include <stdio.h>

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

int main() {
    int c, d;
    printf("Digite o valor de c: ");
    scanf("%d", &c);
    printf("Digite o valor de d: ");
    scanf("%d", &d);
    printf("Soma de c e d: %d\n", soma(c, d));
    printf("Soma de 5 e 10: %d\n", soma(5, 10));
    return 0;
}
```

**Passagem de parâmetros por valor:** Ao chamar uma função passando variáveis como parâmetros, estamos usando apenas os seus *valores*, que são copiados para as variáveis parâmetros da função. Os valores dos parâmetros na chamada da função não são afetados por alterações dentro da função.

## 2.5 Exemplo de Passagem por Valor

```
#include <stdio.h>

int soma(int x, int y) {
    x = x + 1;  // Modifica apenas a copia local
    y = y + 1;  // Modifica apenas a copia local
```

```
    return (x + y);
}

int main() {
    int a = 10, b = 5;
    printf("Soma de a e b: %d\n", a + b);         // 15
    printf("Soma de x e y: %d\n", soma(a, b));    // 17
    printf("a: %d\n", a);  // a continua sendo 10
    printf("b: %d\n", b);  // b continua sendo 5
    return 0;
}
```

As modificações em `x` e `y` dentro da função não afetam `a` e `b` no `main`, pois a função trabalha com cópias dos valores.

# 3 Procedimentos

Um **procedimento** é uma função que não retorna nenhum valor. Procedimentos são indicados pela palavra-chave especial `void`.

```
// Funcao que retorna um valor
int soma(int a, int b) {
    return a + b;
}

// Procedimento que apenas realiza uma acao
void imprime_soma(int a, int b) {
    printf("Soma: %d\n", a + b);
}
```

## 3.1 A Palavra-chave void

Em C, a palavra `void` representa a "ausência" de algum objeto. Pode ser usada para:

- Indicar a ausência de um tipo de retorno:

```
void imprime_numero(int n) {
    printf("Numero: %d\n", n);
}
```

- Explicitar a ausência de parâmetros:

```
int obtem_numero(void) {
    int n;
    scanf("%d", &n);
    return n;
}
```

# 4 A Função main

O programa principal é uma função especial chamada `main` com tipo de retorno `int`. A função `main`:

- É invocada automaticamente pelo sistema operacional quando o programa é executado.
- Deve retornar 0 (zero) caso tenha funcionado corretamente.
- Deve retornar qualquer outro valor caso tenha ocorrido um erro.

```
int main() {
    float dividendo, divisor;
    scanf("%f␣%f", &dividendo, &divisor);
    if (divisor == 0.0) {
        printf("Divisao␣por␣zero!\n");
        return 1;  // Indica erro
    } else {
        printf("Divisao:␣%f\n", dividendo / divisor);
        return 0;  // Indica sucesso
    }
}
```

# 5 Protótipos de Funções

Considere o seguinte código:

```
#include <stdio.h>

int main() {
    float a = 0, b = 5;
    printf("%f\n", soma(a, b));  // ERRO!
    return 0;
}

float soma(float op1, float op2) {
    return op1 + op2;
}
```

Este código resulta em erro de compilação porque o compilador não conhece a função `soma` quando ela é chamada em `main`. O compilador lê o código de cima para baixo, e quando encontra a chamada `soma(a, b)`, ele ainda não viu a definição da função.

## 5.1 Solução: Declarar Protótipos

Para organizar melhor um programa e implementar funções em partes distintas do arquivo, são utilizados **protótipos de funções**. Um protótipo corresponde à primeira linha da declaração de uma função, terminando com ponto e vírgula:

```
tipo nome(tipo parametro1, ..., tipo parametroN);
```

O protótipo de uma função deve vir sempre *antes* do seu uso. A sua definição completa pode aparecer em qualquer lugar do programa. É comum colocar os protótipos de funções no início do arquivo.

```
#include <stdio.h>

float soma(float op1, float op2);  // Prototipo

int main() {
    float a = 0, b = 5;
    printf("%f\n", soma(a, b));  // OK!
    return 0;
}

float soma(float op1, float op2) {  // Definicao
    return (op1 + op2);
}
```

Agora o código compila corretamente, pois o compilador conhece a assinatura da função soma antes de ela ser usada.

## 5.2 Múltiplos Protótipos

Podemos declarar vários protótipos no início do arquivo:

```
#include <stdio.h>

float soma(float op1, float op2);
float subt(float op1, float op2);
float mult(float op1, float op2);

int main() {
    float a = 10, b = 5;
    printf("Soma:␣%f\n", soma(a, b));
    printf("Subtracao:␣%f\n", subt(a, b));
    printf("Multiplicacao:␣%f\n", mult(a, b));
    return 0;
}

float soma(float op1, float op2) {
```

```
    return (op1 + op2);
}

float subt(float op1, float op2) {
    return (op1 - op2);
}

float mult(float op1, float op2) {
    return (op1 * op2);
}
```

# 6 Escopo de Variáveis

O **escopo** de uma variável determina onde ela pode ser acessada no programa. Em C, existem dois tipos principais de variáveis:

## 6.1 Variáveis Locais

Uma variável é chamada **local** se ela foi declarada dentro de uma função.
Características:

- A variável existe somente dentro da função onde foi declarada.
- Quando a execução da função termina, a variável deixa de existir.
- Parâmetros de função são variáveis locais.
- Outras funções não podem acessar essa variável.

## 6.2 Variáveis Globais

Uma variável é chamada **global** se ela for declarada fora de qualquer função.
Características:

- A variável é visível em todas as funções do programa.
- Qualquer função pode modificar a variável.
- A variável existe durante toda a execução do programa.
- Deve ser usada apenas em casos muito especiais.

**Boa prática:** Evite usar variáveis globais sempre que possível. Elas tornam o código mais difícil de entender e manter, pois qualquer função pode modificá-las, dificultando o rastreamento de bugs.

## 6.3 Estrutura Básica de um Programa

```
#include <stdio.h>

// Prototipos de funcoes
int funcao1(int x);
void funcao2(float y);

// Declaracao de variaveis globais (evite!)
int global;

int main() {
    // Declaracao de variaveis locais
    int local_main;

    // Comandos
    // ...

    return 0;
}

int funcao1(int x) {
    // Declaracao de variaveis locais
    int local_funcao1;

    // Comandos
    // ...

    return local_funcao1;
}

void funcao2(float y) {
    // Declaracao de variaveis locais
    float local_funcao2;

    // Comandos
    // ...
}
```

## 6.4 Exemplo de Escopo

```
#include <stdio.h>

void funcao_a(void);
int funcao_b(int local_b);

int global;  // Variavel global

int main() {
    int local_main;
    /* Neste ponto sao visiveis: global, local_main */
}

void funcao_a(void) {
    int local_a;
    /* Neste ponto sao visiveis: global, local_a */
}

int funcao_b(int local_b) {
    int local_c;
    /* Neste ponto sao visiveis: global, local_b, local_c */
}
```

## 6.5 Escondendo Variáveis Globais

É possível declarar variáveis locais com o mesmo nome de variáveis globais. Nesta situação, a variável local "esconde" a variável global dentro da função:

```
#include <stdio.h>

int x;  // Variavel global

void fun1() {
    printf("\n%d", x);  // Usa a variavel global
}

void fun2() {
    int x;  // Variavel local esconde a global
    printf("\n%d", x);  // Usa a variavel local
}

int main() {
    x = 10;
    fun1();  // Imprime 10 (variavel global)
    fun2();  // Imprime um valor indefinido (variavel local nao inicializada)
    return 0;
```

```
}
```

# 7 Passagem de Parâmetros

Considere o seguinte código que tenta trocar os valores de duas variáveis:

```
void trocar_valores(int x, int y) {
    int aux;
    aux = x;
    x = y;
    y = aux;
}

int main() {
    int x = 4, y = 5;
    trocar_valores(x, y);
    printf("x=%d,␣y=%d\n", x, y);  // Imprime x=4, y=5
    return 0;
}
```

Por que não funcionou a troca? Porque os parâmetros foram passados **por valor**. As modificações ocorreram apenas nas cópias locais dos parâmetros dentro da função, não nas variáveis originais.

Os parâmetros de função podem ser passados de dois modos:

## 7.1 Passagem por Valor

- Apenas o resultado da expressão é passado para a função.
- Pode ser qualquer expressão: constantes, somas, variáveis, etc.
- Os valores são copiados para os parâmetros formais.
- **Não alteram** o valor das variáveis originais passadas.
- É o modo padrão em C.

```
int obter_quadrado(int x) {
    return x * x;
}

int main() {
    int x = 2;
    printf("Quadrado␣de␣%d␣eh␣%d\n", x, obter_quadrado(x));
    printf("Quadrado␣de␣%d␣eh␣%d\n", 3, obter_quadrado(3));
```

```
    return 0;
}
```

# 8 Exercícios

1. Escreva uma função que leia do teclado o número de questões de uma prova e o valor de cada uma das questões. A função deve retornar a nota da prova.

2. Escreva uma função que receba como parâmetro um número de provas e leia do teclado o número de questões e os valores das questões de cada prova. A função deve retornar a média das provas.

3. Escreva um programa que leia do teclado um número de provas dadas em um semestre, o número de alunos matriculados e o número de questões e os valores das questões de cada prova de cada aluno. O programa deve imprimir a razão entre a média das notas dos alunos que tiraram pelo menos 5 e a média das notas dos alunos que tiraram abaixo de 5.

4. Escreva uma função que calcule o máximo divisor comum (MDC) de dois números usando o algoritmo de Euclides. O algoritmo baseia-se no fato de que MDC$(a, b)$ = MDC$(b, a \mod b)$ até que $b = 0$.

5. Crie uma função que retorne o $n$-ésimo termo da sequência de Fibonacci.