

Fundamentos de Programação

Funções e Menus



JESUÍTAS BRASIL



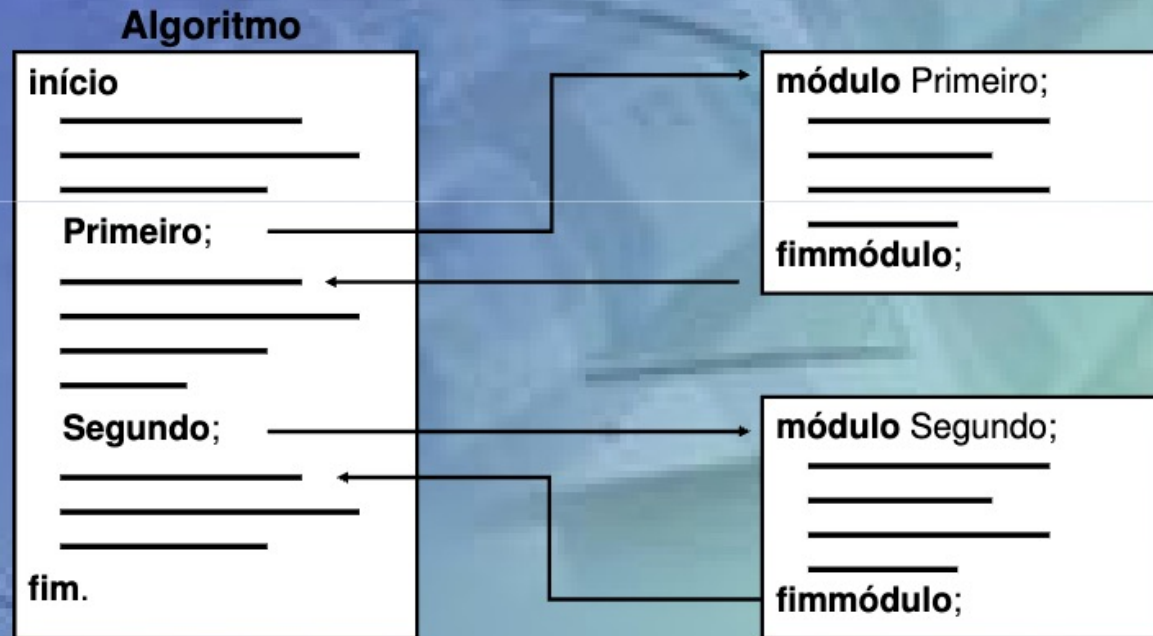
Somos infinitas possibilidades

Funções

- Também conhecidas como métodos ou procedimentos
- Organização do código fonte em blocos padronizados
- Reaproveitamento de códigos executados de forma idêntica em diversas partes do programa

Modularização de Algoritmos

◆ Esquema de Chamada :



Estrutura de uma Função

```
// Função para calcular o fatorial de um número inteiro
int funcao(lista de parametros) {
    // conjunto de operações
    return valor;
}
```

Função básica

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

Declaração e
implementação da função.
Ela só será executada
quando for chamada

```
int main() {  
    int num = 5; // Número para calcular o fatorial  
    int fat = fatorial(num);  
    printf("O fatorial de %d é: %d\n", num, fat);  
    return 0;  
}
```

Chamada da função. Ela é
executada nesse momento,
podendo ser chamada
diversas vezes

Função com parâmetros

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

```
// Função para calcular a soma de dois números inteiros
```

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

Parâmetros são dados enviados pelo chamador para dentro da função, e podem ser usados livremente dentro dela

```
int main() {
```

```
    int num1 = 5; // Primeiro número
```

```
    int num2 = 3; // Segundo número
```

```
    int resultado = soma(num1, num2);
```

```
    printf("A soma de %d e %d é: %d\n", num1, num2, resultado);
```

```
    return 0;
```

```
}
```

Em C, a nossa main é vista como a função principal do programa

Função com retorno

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

```
// Função para calcular a soma de dois números inteiros
```

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

A função soma os valores dos dois parâmetros e retorna o resultado para o chamador

```
int main() {
```

```
    int num1 = 5; // Primeiro número
```

```
    int num2 = 3; // Segundo número
```

```
    int resultado = soma(num1, num2);
```

```
    printf("A soma de %d e %d é: %d\n", num1, num2, resultado);
```

```
    return 0;
```

```
}
```

Na chamada da função, são passados os dois valores e o retorno é armazenado em uma variável

Função sem retorno

```
// Função para imprimir uma mensagem
void imprimir_mensagem() {
    printf("Olá! Esta é uma mensagem da função.\n");
}

int main() {
    // Chamando a função para imprimir a mensagem
    imprimir_mensagem();

    return 0;
}
```


Variáveis globais

```
#include <stdio.h>

// Variável global
int contador = 0;
// Função que incrementa o contador global
void incrementar_contador() {
    contador++;
}

int main() {
    // Chamando a função para incrementar o contador global
    incrementar_contador();

    // Imprimindo o valor do contador global
    printf("Valor do contador: %d\n", contador);

    return 0;
}
```

Variáveis locais

Declaração de variáveis dentro da função

```
void função(int n){  
    int x;  
}
```

Acesso a variáveis globais e locais

- ▶ Variável definida em uma função não é acessível por outra

Passagem de parâmetro por cópia

- ▶ Alteração do parâmetro não interfere no valor original

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

Só existe
localmente na
função

```
int main(){  
    int fat;  
    fat = fatorial(5);  
    printf("Fatorial de 5 é: %d",fat);  
    fat = fatorial(10);  
    printf("Fatorial de 10 é: %d",fat);  
    return 0;  
}
```

Só existe
localmente na
main

Mas e com Vetores???



```
#include <stdio.h>

#define TAMANHO_VETOR 5

// Função para imprimir os elementos de um vetor
void imprimir_vetor(int vetor[]) {
    printf("Vetor: ");
    for (int i = 0; i < TAMANHO_VETOR; i++) {
        printf("%d ", vetor[i]);
    }
    printf("\n");
}

int main() {
    int vetor[TAMANHO_VETOR] = {1, 2, 3, 4, 5};

    // Chamando a função para imprimir o vetor
    imprimir_vetor(vetor);

    return 0;
}
```

É possível também que o parâmetro seja um vetor de tamanho dinâmico, para isso é necessário que uma variável inteira guardando o tamanho do vetor também seja passada por parâmetro para a função

```
#include <stdio.h>

void imprimirVetor(int n, int v[n]) {
    int i;
    printf("Vetor: ");
    for(i=0; i<=n-1; i++) {
        printf("%d ", v[i]);
    }
}

int main() {
    int vet[10], i;
    for(i=0; i<=9; i++) {
        printf("Informe o num da pos %d: ", i);
        scanf("%d", &vet[i]);
    }
    imprimirVetor(10, vet);
}
```

Mas e cuidado com Vetores



```
// Função para dobrar os elementos de um vetor
void dobrar_elementos(int vetor[], int tamanho) {
    for (int i = 0; i < tamanho; i++) {
        vetor[i] *= 2;
    }
}

int main() {
    int vetor[5] = {1, 2, 3, 4, 5};
    int tamanho = 5;

    // Chamando a função para dobrar os elementos do vetor
    dobrar_elementos(vetor, tamanho);
    // Imprimindo o vetor modificado
    printf("Vetor dobrado: ");
    for (int i = 0; i < tamanho; i++) {
        printf("%d ", vetor[i]);
    }

    return 0;
}
```


1. Faça um procedimento (função com retorno vazio) que recebe um número inteiro imprima se ele é par ou ímpar. No programa principal leia indeterminados números positivos (até que seja informado um número negativo) e informe para cada um desse número se ele é par ou ímpar.
2. Faça um programa que leia um número N (inteiro) e calcule e imprima o fatorial de N. Para isso desenvolva uma função para calcular o fatorial de um número.
3. Faça um programa em C que leia um número N (inteiro) e calcule e imprima a quantidade de dígitos que o número N possui. Para isso desenvolva uma função para calcular a quantidade de dígitos de um número.
4. Faça uma função que verifique se um numero é primo. A função deve retornar um valor lógico (0 – falso e 1 – verdadeiro). No programa principal leia indeterminados números inteiros positivos (até que o usuário digite -1) e escreva se cada um dos números é primo ou não.

Passagem por parâmetros e por referência

- Existem dois tipos de passagem de parâmetros para uma função:
 - Passagem **por valor**
 - Passagem **por referência**
- Para entender a diferença entre esses tipos, é importante compreender o conceito de **escopo de variáveis**

Passagem por parâmetros e por referência

- Variáveis **Globais**:

- Declaradas fora de qualquer função (incluindo `main`)
 - no início do programa
- São visíveis em qualquer parte do programa

- Variáveis **Locais**:

- Declaradas dentro de funções
- São visíveis apenas dentro do escopo a que pertencem
- “Desaparecem” quando a função encerra sua execução

- **Conflito**:

- Vale a variável local

Passagem por parâmetros e por referência

- Geralmente, a passagem de parâmetros em C é realizada **por valor**:
 - Alterações feitas nos parâmetros recebidos não se refletem nos valores passados como argumentos
 - O valor do argumento é *copiado* durante a chamada da função
 - Para ilustrar, façamos uma pequena modificação no exemplo anterior da Nota 2...

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

```
// Função que soma dois números e retorna o resultado
```

```
int soma(int a, int b) {
```

```
    return a + b; // Retorna a soma dos parâmetros
```

```
}
```

```
int main() {
```

```
    int x = 5;
```

```
    int y = 10;
```

```
    // Chamando a função soma com x e y como argumentos
```

```
    int resultado = soma(x, y);
```

```
    printf("A soma de %d e %d é %d.\n", x, y, resultado);
```

```
    return 0;
```

```
}
```

Passagem por parâmetros e por referência

- Passagem **por referência**

- Alterações feitas nos parâmetros recebidos são refletidas nos valores passados como argumentos
- Para entender este processo em linguagem C, é preciso antes compreender o conceito de ponteiro
 - tópico avançado no curso...

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

```
// Função que incrementa o valor de dois números
```

```
void incrementa(int *a, int *b) {
```

```
    (*a)++; // Incrementa o valor apontado por a
```

```
    (*b)++; // Incrementa o valor apontado por b
```

```
}
```

```
int main() {
```

```
    int x = 5;
```

```
    int y = 10;
```

```
    // Imprime os valores de x e y antes da chamada da função
```

```
    printf("Antes: x = %d, y = %d\n", x, y);
```

```
    // Chamando a função incrementa com os endereços de x e y
```

```
    incrementa(&x, &y);
```

```
    // Imprime os valores de x e y após a chamada da função
```

```
    printf("Depois: x = %d, y = %d\n", x, y);
```

```
    return 0;
```

```
}
```

```
#include <math.h>
```

```
int segundo_grau(float a, float b, float c, float *x1, float *x2) {
```

```
    float delta = b*b - 4*a*c;
```

```
    if (delta < 0)
```

```
        return -1;
```

```
    *x1 = (-b + sqrt(delta)) / (2*a);
```

```
    *x2 = (-b - sqrt(delta)) / (2*a);
```

```
    if (delta > 0)
```

```
        return 1;
```

```
    return 0;
```

```
}
```



```
int main() {  
    float a, b, c, r1, r2, tem_raiz;  
    printf("Entre com os coeficientes a, b e c: ");  
    scanf("%f %f %f", &a, &b, &c);  
  
    tem_raiz = segundo_grau(a, b, c, &r1, &r2);  
    if (tem_raiz < 0)  
        printf("Equação não tem raízes reais.\n");  
    else if (tem_raiz == 0)  
        printf("Equação tem somente uma raiz: %f.\n", r1);  
    else  
        printf("Equação tem duas raízes: %f e %f.\n", r1, r2);  
    return 0;  
}
```

Passagem por parâmetros e por referência

Exercício 1: Escreva um programa em C que possua uma função que troque os valores de duas variáveis usando passagem de parâmetros por referência.

Exercício 2: Escreva uma função sem retorno em C para somar dois valores e armazenar o resultado em uma terceira variável. A terceira variável não pode ser definida na função e precisa ser passada como referência.

Protótipo de função

Em C, a regra de que as funções precisam ser declaradas antes de serem usadas tem a ver com a maneira como o compilador processa o código. C é uma linguagem de programação compilada, o que significa que o compilador precisa conhecer a existência e o tipo de dados de todas as funções e variáveis antes de utilizá-las no código. Isso ajuda o compilador a fazer a verificação de tipos e a alocar a quantidade correta de memória para os argumentos de função quando uma função é chamada.

Protótipo de função

Quando você declara ou define uma função antes de main, você está informando ao compilador sobre a existência dessa função, seus parâmetros e o tipo de retorno dela. Isso permite que o compilador saiba como lidar com essa função mais tarde no código, mesmo antes de chegar à definição completa da função.

Protótipo de função

```
#include <stdio.h>

void minhaFuncao(int a); // Protótipo de função – declaração

int main() {
    minhaFuncao(5);
    return 0;
}

void minhaFuncao(int a) { // Definição de função
    printf("%d\n", a);
}
```

```
#include <stdio.h>

// Protótipo da função
int quadrado(int n);

int main() {
    int valor = 5;
    int resultado = quadrado(valor); // Chamada da função antes de sua definição completa
    printf("O quadrado de %d é %d.\n", valor, resultado);
    return 0;
}

// Definição da função, conforme declarado pelo protótipo
int quadrado(int n) {
    return n * n;
}
```

Exercício 3: Desenvolva um programa em C que utilize protótipos de funções para implementar operações básicas de uma calculadora (adição, subtração, multiplicação e divisão). O programa deve permitir ao usuário escolher a operação desejada e inserir dois números, exibindo o resultado da operação escolhida.

```
// Declaração das funções auxiliares
```

```
int soma(int x, int y);
```

```
int multiplicacao(int x, int y);
```

```
// Função que chama outras funções
```

```
void calcularOperacoes(int a, int b) {
```

```
    int resultadoSoma = soma(a, b);
```

```
    int resultadoMultiplicacao = multiplicacao(a, b);
```

```
    printf("Soma de %d e %d é: %d\n", a, b, resultadoSoma);
```

```
    printf("Multiplicação de %d e %d é: %d\n", a, b, resultadoMultiplicacao);
```

```
}
```

```
// Definição da função soma
```

```
int soma(int x, int y) {
```

```
    return x + y;
```

```
}
```

```
// Definição da função multiplicacao
```

```
int multiplicacao(int x, int y) {
```

```
    return x * y;
```

```
}
```

**PODEMOS CHAMAR
VÁRIAS FUNÇÕES DENTRO
DE OUTRAS COM OU SEM
PROTÓTIPOS DE FUNÇÕES**

- Ativação e captura do resultado de uma função pode ser:
 - Por atribuição direta do retorno a uma variável do mesmo tipo
 - Por uso do retorno dentro de uma expressão
- Exemplo (H é **real**, A e Y são **inteiros** ou **reais**):

$H = \text{sqrt}(A + \text{pow}(Y, 2));$

atribuição direta uso em expressão

- Dados dois números N e K , calcular a Combinação:

$$C = \frac{N!}{K!(N-K)!}$$

- Se existisse uma função **fat**(X) que calculasse o fatorial de um dado X , o cálculo acima ficaria:

$$C = \mathbf{fat}(N) / (\mathbf{fat}(K) * \mathbf{fat}(N-K))$$

Fundamentos de Programação

Matrizes



JESUÍTAS BRASIL



Somos infinitas possibilidades

	Aluno		Nota1		Nota2		Nota3		Nota4		Média
0	Pedro	0	5.6	0	6.0	0	7.3	0	5.6	0	6.1
1	Ana	1	10	1	4.0	1	5.0	1	7.3	1	6.6
2	Luiz	2	4.5	2	2.0	2	5.5	2	1.0	2	3.3
...
48	Matheus	48	7.2	48	6.6	48	8.1	48	8.8	48	7.7
49	Andre	49	6.0	49	9.0	49	7.3	49	4.5	49	6.6



JESUÍTAS BRASIL

UNISINOS

Somos infinitas possibilidades

E se tivermos que armazenar 100 notas?

	Aluno		Nota1		Nota2		Nota3				Nota99		Nota100		Média	
0	Pedro	0	5.6	0	6.0	0	7.3				0	7.3	0	5.6	0	6.1
1	Ana	1	10	1	4.0	1	5.0				1	5.0	1	7.3	1	6.6
2	Luiz	2	4.5	2	2.0	2	5.5				2	5.5	2	1.0	2	3.3
...
48	Matheus	48	7.2	48	6.6	48	8.1				48	8.1	48	8.8	48	7.7
49	Andre	49	6.0	49	9.0	49	7.3				49	7.3	49	4.5	49	6.6

Criaremos 100 vetores com 100 nomes diferentes?



JESUÍTAS BRASIL



UNISINOS

Somos infinitas possibilidades

Uma solução mais eficaz para resolver o problema é o uso de **matrizes**:

Aluno		0	1	2	...	98	99	Média		
0	Pedro	0	5.6	6.0	7.3	...	7.3	5.6	0	6.1
1	Ana	1	10	4.0	5.0	...	5.0	7.3	1	6.6
2	Luiz	2	4.5	2.0	5.5	...	5.5	1.0	2	3.3
...
48	Matheus	48	7.2	6.6	8.1	...	8.1	8.8	48	7.7
49	Andre	49	6.0	9.0	7.3	...	7.3	4.5	49	6.6



Uma matriz é uma estrutura de dados de **variáveis de mesmo tipo**, acessíveis com um **único nome** e armazenados contiguamente na memória.

A individualização de cada variável de um vetor é feita através do uso de **índices**.

Os **Vetores** são matrizes de uma só dimensão.

```
int Vetor[5];    // declara um vetor de 5 posições  
  
int Matriz[5][3]; // declara uma matriz de 5 linhas e 3 colunas
```



JESUÍTAS BRASIL



Somos infinitas possibilidades

Uma matriz é uma estrutura de dados de **variáveis de mesmo tipo**, acessíveis com um **único nome** e armazenados contiguamente na memória.

A individualização de cada variável de um vetor é feita através do uso de **índices**.

```
int Vetor[5]; // declara um vetor de 5 posições

int Matriz[5][3]; // declara uma matriz de 5 linhas e 3 colunas

Vetor[0] = 9; // coloca 9 na primeira posição do vetor
Vetor[4] = 30 // coloca 30 na última posição do vetor

Matriz[0][1] = 15; // coloca 15 na célula que está na primeira linha
                  // e na segunda coluna da matriz
```



Podemos também definir constantes para o tamanho da matriz.

```
#define NLIN 10  
#define NCOL 10  
  
int Matriz[NLIN][NCOL];
```



Preenchendo uma matriz

```
for(i=0; i < NLIN; i++){  
    for(j=0; j < NCOL; j++){  
        Matriz[i][j] = 30;  
    }  
}
```



```
//imprimir o elemento da linha 3 e coluna 10 da matriz notas  
printf("%lf", notas[3][10]);  
  
//multiplica a posição (i, j) da matriz mat por 5;  
mat[i][j] = mat[i][j] * 5;
```



```
// Declaração de uma matriz 3x4, inicializada com zeros
int matriz[3][4] = {0};

// Inicialização explícita da matriz com valores específicos
int matrizExemplo[3][4] = {
    {1, 2, 3, 4}, // Primeira linha
    {5, 6, 7, 8}, // Segunda linha
    {9, 10, 11, 12} // Terceira linha
};
```



```
// Imprimindo os valores da matrizExemplo
for(int i = 0; i < 3; i++) {
    for(int j = 0; j < 4; j++) {
        printf("%d ", matrizExemplo[i][j]);
    }
    printf("\n"); // Quebra de linha para cada nova linha da matriz
}
```



Lendo como input do teclado

```
int matriz[3][3];

printf("Digite os valores para uma matriz 3x3:\n");

// Loop para ler os valores da matriz do teclado
for(int i = 0; i < 3; i++) {
    for(int j = 0; j < 3; j++) {
        printf("Elemento [%d][%d]: ", i, j);
        scanf("%d", &matriz[i][j]);
    }
}
```



Exercício 1 – Faça um programa que leia os elementos de uma matriz dados pelo usuário e imprima ela na tela

Exercício 2 – Faça um programa que tenha duas matrizes 3X3 e calcule a soma das duas. Imprima o resultado na tela



JESUÍTAS BRASIL



Somos infinitas possibilidades

Dada uma matriz (4×5), calcular a soma de todos os elementos da matriz. Calcular também o somatório dos elementos de cada linha da matriz, armazenando o somatório em um vetor.

SOMALINHA

MAT

1	2	3	4	5
0	-1	0	-3	1
2	-2	-2	2	0
0	0	6	0	0

15
-3
0
6



JESUÍTAS BRASIL

UNISINOS

Somos infinitas possibilidades

Inicialização de matrizes pt 2

Inicializando na declaração. Processo semelhante à inicialização de vetores.

```
1  int matriz[3][4] = { {10, 20, 30, 40},  
2                        {50, 60, 70, 80},  
3                        {90, 11, 22, 33} };
```

Mas podemos fazer também:

```
1  int matriz[3][4] = { 10, 20, 30, 40,  
2                        50, 60, 70, 80,  
3                        90, 11, 22, 33 };
```

Ou ainda:

```
1  int matriz[3][4] = { 10, 20, 30, 40, 50, 60, 70, 80, 90, 11, 22, 33 };
```

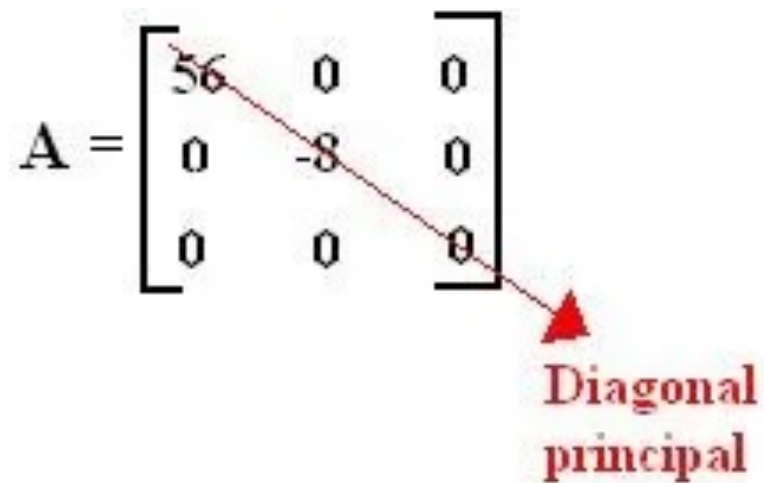


Funções e Matrizes

```
void imprimirMatriz(int matriz[3][3], int n, int m)
{
    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < m; ++j){
            printf("%d ", matriz[i][j]);
            printf("\n");
        }
    }
}
```



Exercício 4 – Faça um programa em C que implementa um função que recebe uma matriz e imprime a sua diagonal principal.

$$A = \begin{bmatrix} 56 & 0 & 0 \\ 0 & -8 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$


Diagonal principal



JESUÍTAS BRASIL



Somos infinitas possibilidades

DESAFIO! Com uma matriz 3x3, estruturas de repetição e condicionais (if – else), como podemos implementar um jogo da velha em C?



JESUÍTAS BRASIL



Somos infinitas possibilidades