

Lógica de Programação: notas de aula

Prof. Jonatha Costa

2025

Organização

1 Blocos de programação - C

Funções

Visão geral de definições e chamadas

Tabela-resumo: tipos de subfunções e chamadas

2 Modularização

3 Exercícios para explorar multiplas soluções

Objetivo da aula

- Estudar a construção de funções para utilização dentro do código principal e utilização de bibliotecas criadas pelo usuário.

Script

Quais elementos de programação estão presentes no *script*?

Exemplo

```
#include<stdio.h>          // biblioteca
int main()                  // Função principal
{ printf("Oi!");           // comando da biblioteca
}
```

- É possível criar bibliotecas e comandos(rotinas) próprias?
- Como criar uma função?
- Como criar uma biblioteca?

O que são funções em C?

- São agrupamentos de instruções sob um nome identificador.
- Facilitam o reaproveitamento de código.
- Aumentam a legibilidade e manutenção.

Funções

Sintaxe

```
tipo_saída nome_da_funcao (tipo_entrada  
var_entrada)  
{  
<comandos>;  
}
```

Exemplo

```
int soma(int var1, int var2)  
{  
int res;  
res = var1 + var2;  
return res;  
}
```

No arquivo principal, essa função pode ser definida e declarada em bloco único, ao início de código seguindo a estrutura:

- ① Declaração e definição da função;
- ② Código main.

Ou ainda, essa função pode ser definida ao início de código, antes do main(), seguido-se da definição da função:

- ① Declaração da função;
- ② Código main;
- ③ Definição da função.

Funções: declaração e definição juntas

Exemplo de função no próprio código

```
#include<stdio.h>
int soma(int v1,int v2) // Declarando e definindo a função
{ int res=v1 + v2;
return res;
}
```

```
int main( ) // Programa principal
{    int resultado, a, b;
printf("Digite um numero a:");
scanf("%d", &a);
printf("Digite um numero b:");
scanf("%d", &b);
resultado=soma(a , b); // Chamando a função
printf("Soma = %d", resultado);
}
```

Funções em modo declaração e definição geminados

Perceba que, neste modo, o *script* contém uma função declarada e já definida no topo de código, e que a função é evocada depois pelo código principal.

Estrutura:

- ① Declaração e definição da função;
- ② Código main.

Funções: declaração e definição separadas

Exemplo de função no próprio código

```
#include<stdio.h>
int soma(int v1,int v2); // Declarando a função

int main() // Programa principal
{
    int resultado, a, b;
    printf("Digite um numero a:");
    scanf("%d", &a);
    printf("Digite um numero b:");
    scanf("%d", &b);
    resultado=soma(a , b); // Chamando a função
    printf("Soma = %d", resultado);
}

int soma(int v1,int v2) // Definindo a função
{ int res=v1 + v2;
return res;
}
```

Funções em modo declaração e definição separados

Perceba que, neste outro modo, o *script* contém uma função declarada no topo de código, antes da função *main()*, seguindo-se, então a definição da função correspondente à declaração.

Estrutura:

- ① Declaração da função;
- ② Código main;
- ③ Definição da função.

Estruturas antes ou depois

Note que:

- **Fluxo do compilador:** Em linguagens como C, o compilador lê o código de cima para baixo. Se o programador utilizar uma função antes de defini-la, o compilador retornará um erro por não saber o que fazer com essa função.
- **Pré-declaração:** A declaração antes do "main" diz ao compilador o que ele precisa saber sobre a função, para que ela possa ser utilizada antes da definição completa.
- **Código mais organizado:** Colocar a declaração da função no início permite que o programador mantenha a função main na parte superior do código, tornando-a mais fácil de encontrar e ler. A declaração também torna o **código mais modular**, visto que o programador pode definir funções em qualquer lugar, desde que o compilador saiba de sua existência.

Tipos de Funções e seus Relacionamentos

Os principais tipos de subfunções em linguagem C e como elas se relacionam com o programa principal (`main()`), compreendem:

- Diferentes formas de definir e chamar funções;
- Tipos de **retorno**: `void`, `int`, `float`;
- Formas de passagem de parâmetros: por valor, por ponteiro e por vetor;
- Funções que operam com alocação dinâmica;

Note que compreender esses formatos é essencial para modularizar programas, reutilizar código e trabalhar com estruturas de dados em projetos reais em C.

Subfunção 01: void func(void)

Características:

- Não recebe parâmetros de entrada.
- Não retorna valor ao programa principal.

Definição:

```
void saudacao(void) {  
    printf("Bem-vindo!\n");  
}
```

Chamada no main():

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

Subfunção 02: int/float func(void)

Características:

- Não recebe parâmetros de entrada.
- Retorna um valor ao programa principal.

Definição:

```
int obterAno(void) {  
    return 2025;  
}
```

Chamada no main():

```
int main() {  
    int ano = obterAno();  
    printf("%d\n", ano);  
    return 0;  
}
```

Subfunção 03: void/int func(int)

Características:

- Recebe parâmetros como entrada.
- Pode retornar ou não um valor.

Definição:

```
void exibeQuadrado(int n) {  
    printf("%d\n", n * n);  
}  
  
int dobro(int x) {  
    return 2 * x;  
}
```

Chamada no main():

```
1 int main() {  
2     exibeQuadrado(5);  
3  
4     int resultado = dobro(7);  
5     printf("%d\n", resultado);  
6  
7     return 0;  
8 }
```

Subfunção 04: void func(int in[], int out[])

Características:

- Recebe vetores como parâmetros.
- Altera vetor de saída por referência.

Definição:

```
void quadrado(int in[], int out
    [], int n) {
for (int i = 0; i < n; i++) {
out[i] = in[i] * in[i];
}
}
```

Chamada no main():

```
1 int main() {
2     int a[] = {1, 2, 3};
3     int b[3];
4
5     quadrado(a, b, 3);
6
7     for (int i = 0; i < 3; i++)
8         printf("%d ", b[i]);
9
10    return 0;
11 }
```

Subfunção 05: void func(int m[] [COL])

Características:

- A função recebe uma matriz como argumento¹.
- Altera os dados da matriz original por referência.

Definição:

```
void dobrarMatriz(int m[] [3]) {
    int lin=2,col=3;
    for (int i = 0; i < lin; i++) {
        for (int j = 0; j < col; j++) {
            m[i][j] *= 2;
        }
    }
}
```

Chamada no main():

```
1 int main() {
2     int mat[2][3] = {
3         {1, 2, 3},
4         {4, 5, 6}};
5     dobrarMatriz(mat);
6     for (int i = 0; i < 2; i++) {
7         for (int j = 0; j < 3; j++) {
8             printf("%d ", mat[i][j]);
9             printf("\n");
10        }
11    }
12 }
```

¹ A segunda dimensão da matriz, no mínimo, precisa estar definida para que o compilador possa calcular os deslocamentos de memória corretamente, pois a linguagem C armazena matrizes em **blocos lineares** de memória.

Tabela-resumo: Tipos de Subfunções em C

Tipos de subfunções em linguagem C

Tipo de função	Definição	Chamada	Características
void func(void)	void saudacao(void)	saudacao();	Sem parâmetros, sem retorno. Executa apenas uma ação.
int func(void)	int obterAno(void)	int x = obterAno();	Sem parâmetros, retorna um valor escalar.
void func(int)	void imprime(int x)	imprime(5);	Recebe um parâmetro, sem retorno.
int func(int)	int dobro(int x)	int y = dobro(5);	Recebe valor, retorna resultado. Passagem por valor.
void func(int[], int)	void dobraVetor(int v[], int n)	dobraVetor(v, 3);	Passagem de vetores (ponteiros). Permite modificar os dados originais.
void func(int[] [COL])	void dobraMatriz(int m[][] [3])	dobraMatriz(mat);	Passagem de matrizes 2D. Segunda dimensão deve ser fixa. Permite alteração do conteúdo.

Exercícios utilizando funções

Veja o slide 33 – Exercícios para explorar multiplas soluções.

Organização

1 Blocos de programação - C

2 Modularização

Conceitos de modularização

Conceitos distintos de programação

3 Exercícios para explorar multiplas soluções

O que é Modularização?

- Estratégia para dividir um programa em múltiplos **arquivos**.
- Cada módulo tem uma responsabilidade específica.
- Facilita organização, testes e trabalho em equipe.

Modularizando

- As funções podem ser definidas num arquivo biblioteca. Desse modo, o programador pode evocar, no programa **main**, as funções por ele definidas;
- Faz-se necessário, entretanto:

Configurar os arquivos de blocos

- ① Criar o arquivo “.c” (**ScriptDesejado.c**) contendo o *script* desejado;
- ② Criar um arquivo de biblioteca com a extensão ’h’ (**biblioteca.h**) contendo o nome do arquivo “.c”;
- ③ Incluir no cabeçalho do programa *main* o arquivo **biblioteca.h** e evocar, no *main.c*, o programa (ou função) declarado(a) na biblioteca.

Esse método é utilizado para partitionar um programa grande em blocos menores a fim de melhorar o controle das partes e interações.

Programa em módulos/blocos

“Biblioteca.h”

```
:  
void Aula6Ex1();  
:  
“main.c”  
#include<stdio.h>  
#include  
“Biblioteca.h”  
main()  
{    Aula6Ex1();  
:  
}
```

Script_desejado.c

```
void Aula6Ex1()  
{  
printf("\n*****\n");  
printf("Programando em blocos!");  
printf("\n*****\n");  
}
```

Programa em módulos/blocos - Exemplo 2

“calc.h”

```
int soma(int v1, int v2);
int subtracao (int v1, int v2);
int divisao (int v1, int v2);
int multiplicacao (int v1, int v2);
```

“main.c”

```
#include<stdio.h>
#include “calc.h”
int main()
{
    int res;
    int v1 = 2, v2 = 3;
    res=soma(v1, v2);
    printf("A soma (v1 + v2) vale %d.",res);
}
```

calc.c

```
int soma(int v1,int v2)
{
    return v1+v2;
}
```

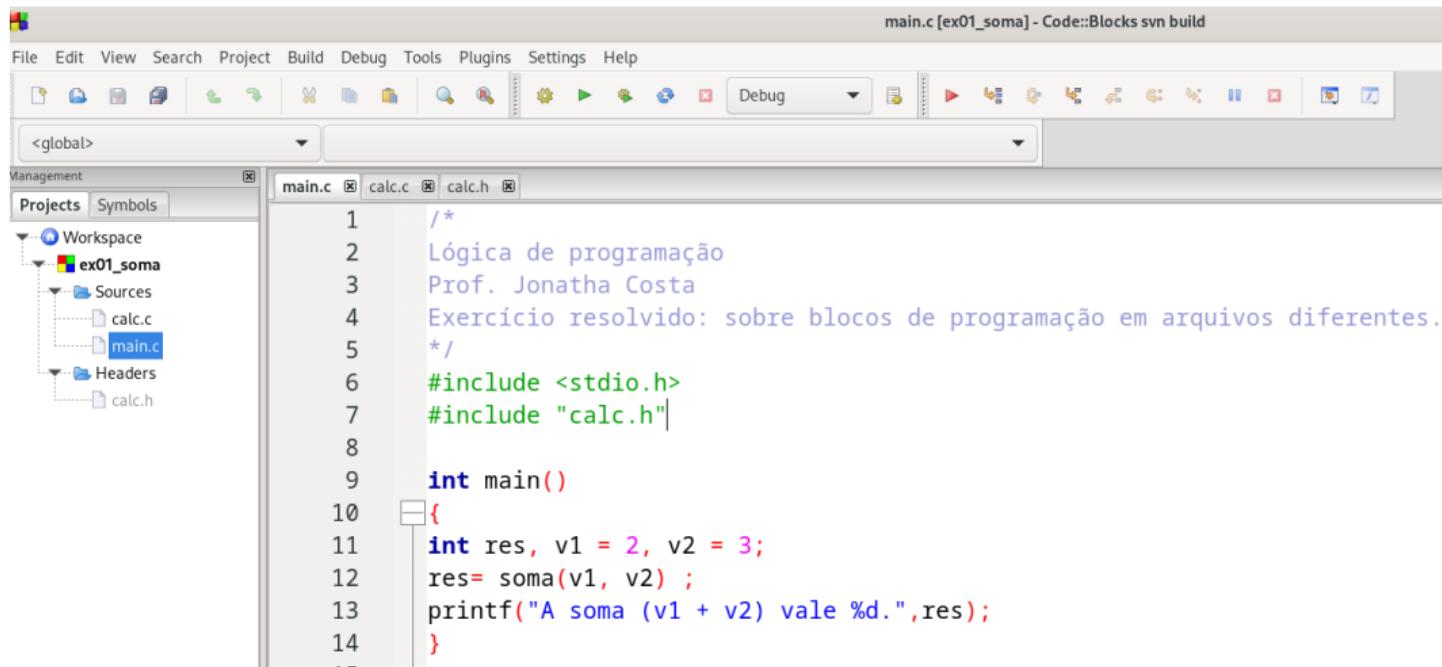
```
int subtracao(int v1,int v2)
{
    return v1-v2;
}
```

```
int multiplicacao(int v1,int v2)
{
    return v1*v2;
}
```

```
int divisao(int v1,int v2)
{
    return v1/v2;
}
```

Programa em módulos/blocos

Figura: Exemplo de programação em blocos utilizando o Code Blocks® IDE



The screenshot shows the Code::Blocks IDE interface. The menu bar includes File, Edit, View, Search, Project, Build, Debug, Tools, Plugins, Settings, and Help. The toolbar has various icons for file operations like Open, Save, and Build. The status bar at the top right says "main.c [ex01_soma] - Code::Blocks svn build". The left sidebar shows the project structure under "Management" and "Projects". The "Projects" tab is selected, showing a workspace named "ex01_soma" containing a "Sources" folder with "calc.c" and "main.c", and a "Headers" folder with "calc.h". The main code editor window displays the "main.c" file with the following content:

```
1  /*
2   * Lógica de programação
3   * Prof. Jonatha Costa
4   * Exercício resolvido: sobre blocos de programação em arquivos diferentes.
5   */
6  #include <stdio.h>
7  #include "calc.h"
8
9  int main()
10 {
11     int res, v1 = 2, v2 = 3;
12     res= soma(v1, v2) ;
13     printf("A soma (v1 + v2) vale %d.",res);
14 }
```

Fonte: AUTOR (2024)

Programa em módulos/blocos

Figura: Exemplo de programação em blocos utilizando o onlinegdb® IDE

The screenshot shows the onlinegdb IDE interface. The top bar includes icons for file operations (New, Open, Save, Share, Beautify), a Run/Debug button, and a Language selector set to C. Below the bar, there are tabs for main.c, calc.h, calc.c, and other files. The main code editor displays the following C code:

```
1 /*
2 Lógica de programação
3 Prof. Jonatha Costa
4 Exercício resolvido: sobre blocos de programação em arquivos diferentes.
5 */
6 #include <stdio.h>
7 #include "calc.h"
8
9 int main()
10 {
11     int res, v1 = 2, v2 = 3;
12     res= soma(v1, v2) ;
13     printf("A soma (v1 + v2) vale %d.",res);
14 }
```

Fonte: AUTOR (2024)

Exercícios utilizando funções

Veja o slide 33 – Exercícios para explorar multiplas soluções.

Script com conceitos distintos

Um mesmo código pode ser produzido com conceitos diferentes.

- ① Rotina dentro do programa principal e no mesmo arquivo (*main.c*);
- ② Rotina fora do programa principal, mas contida no mesmo arquivo (*main.c*);
- ③ Rotina fora do programa principal (*main.c*), mantendo apenas o cabeçalho no arquivo (*main.c*);
- ④ Rotina fora do programa principal (*main.c*); cabeçalho e rotina em arquivos distintos, respectivamente (*rotina.h*) e (*rotina.c*);

Observe com atenção os conceitos e estruturas de cada *script* e apresente as vantagens e desvantagens de cada um.

Script 01

Figura: Rotina contida no arquivo principal e na função *main()*

```
1  /*
2   Lógica de programação
3   Prof. Jonatha Costa
4   Exercício resolvido: Ler 10 números
5   */
6  #include<stdio.h>
7  int main()
8  {
9      int tam_vet=10;
10     int num[tam_vet];
11     for (int i=0;i<tam_vet;i++)
12     {
13         printf("Informe um número (%d / %d): ",i,tam_vet);
14         scanf("%d",&num[i]);
15     }
16 }
17
```

Fonte: AUTOR (2024)

Script 02

Figura: Rotina contida no arquivo principal, porém fora da função *main()*

```
1  /*
2   Lógica de programação
3   Prof. Jonatha Costa
4   Exercício resolvido: Ler 10 números
5   */
6 #include<stdio.h>
7
8 void CarregarVetor(int num[],int tam_vet)
9 {
10    for (int i=0;i<tam_vet;i++)
11    {
12        printf("Informe um número (%d / %d): ",i,tam_vet);
13        scanf("%d",&num[i]);
14    }
15
16 int main()
17 {
18     int tam_vet=10;
19     int num[tam_vet];
20     CarregarVetor(num,tam_vet);
21 }
```

Fonte: AUTOR (2024)

Script 03

(a) Arquivo *main.c*

```
main.c  CarregarVetor0.c
1  /*
2   Lógica de programação
3   Prof. Jonatha Costa
4   Exercício resolvido: Ler 10 números
5   */
6 #include<stdio.h>
7 // Declaração de cabeçalho
8 void CarregarVetor(int num[],int tam_vet);
9 // Programa principal
10 int main()
11 { int tam_vet=10;
12   int num[tam_vet];
13   CarregarVetor(num,tam_vet);
14 }
```

(b) Arquivo *RotCarVetor.C*

```
main.c  CarregarVetor0.c
1  // Rotina de carregar vetor
2 #include<stdio.h>
3 void CarregarVetor(int num[],int tam_vet)
4 {
5   for (int i=0;i<tam_vet;i++)
6   {
7     printf("Informe um número (%d / %d): ",i,tam_vet)
8     scanf("%d",&num[i]);
9   }
10
11 }
```

Figura: Rotina fora do arquivo principal e da função *main()*

Fonte: AUTOR (2024)

Script 04

(a) Arquivo *main.c*

```

main.c RotCarVetor.c RotCarVetor.h
1  /*
2  Lógica de programação
3  Prof. Jonatha Costa
4  Exercício resolvido: Ler 10 números
5  */
6 #include<stdio.h>
7 #include "RotCarVetor.h"
8
9 int main()
10 { int tam_vet=10;
11   int num[tam_vet];
12   CarregarVetor(num,tam_vet);
13 }
14

```

(b) Arquivo *RotCarVetor.C*

```

main.c RotCarVetor.c RotCarVetor.h
1 #include<stdio.h>
2 void CarregarVetor(int num[],int tam_vet)
3 {
4   for (int i=0;i<tam_vet;i++)
5   {printf("Informe um número (%d / %d): ",i,tam_vet);
6    scanf("%d",&num[i]);
7   }
8 }
9

```

(c) Arquivo *RotCarVetor.h*

```

main.c RotCarVetor.c RotCarVetor.h
1 // Declaração de cabeçalho
2 void CarregarVetor(int num[],int tam_vet);
3

```

Figura: Rotina fora do arquivo principal e da função *main()*. Cabeçalhos no arquivo *RotVet.h*

Fonte: AUTOR (2024)

Considerações finais

Apresente as vantagens e desvantagens de cada conceito de programação contida nos *scripts* acima!

Organização

- ① Blocos de programação - C
- ② Modularização
- ③ Exercícios para explorar multiplas soluções

Exercícios para explorar multiplas soluções

Exercícios de fixação para explorar com multiplas soluções.

Bloco 01 - Escreva um programa na linguagem C para:

- ① Ler um número e informar se o número é maior, menor ou igual a 7, 0;
- ② Ler um número e informar se o número par ou ímpar;
- ③ Ler um número e informar se o número é primo ou não;
- ④ Ler um número e informar se o número pertence aos N;

Bloco 02 - Escreva um programa na linguagem C para:

- ① Ler 5 valores, encontrar o maior, o menor e a média utilizando números reais (float).
- ② Ler uma letra e verificar se é uma vogal ou não.
- ③ Leia um número entre 0 e 10, e escreva este número por extenso.
- ④ Elabore um código que receba dois números, a e b tal que $0 \leq a \leq 10$ e $25 \leq b \leq 100$, identifique e informe os valores ímpares de primos contidos nesse intervalo.

Exercícios para explorar multiplas soluções

Bloco 03 - Escreva um programa na linguagem C para calcular e informar:

$$\textcircled{1} \quad z = \sum_{i=1}^{10} x_i$$

$$\textcircled{2} \quad z = \sum_{i=1}^{10} x_i y_i$$

$$\textcircled{3} \quad z = \sum_{i=1}^{10} (\sqrt{i} x_i^2 + y_i^2)$$

Em que $x=\{1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}$ e $y=\{10, 9, 8, 7, 6, 5, 4, 3, 2, 1\}$.

Exercícios para explorar multiplas soluções

Bloco 04 - Escreva um programa na linguagem C para:

- ① Ler uma matriz de 3 por 3, exibí-la e verificar se esta é triangular inferior e informar ao usuário.
- ② Ler e preencher uma matriz de 3 por 3, exibí-la e verificar se esta é triangular inferior, superior ou matriz diagonal e informar ao usuário.
- ③ Escrever um programa que retorne ao usuário o k-ésimo dígito da parte não inteira de π e o valor de π até o dígito k-ésimo dígito. Assuma que π tem apenas 13 dígitos em sua parte não-inteira que o usuário desconhece isto.

Exercícios para explorar multiplas soluções

Bloco 05 - Escreva um programa na linguagem C para:

- ① Ler dez(10) números, ou ‘n’ números conforme escolha do usuário;
- ② Informar o maior, o menor e a média aritmética entre os números;
- ③ Informar quais números são pares, ímpares e primos;
- ④ Calcular a variância e o desvio padrão da série de números;
- ⑤ Reiniciar o processo até que o usuário informe que deseja encerrá-lo.

Nota:

Trabalhe com o máximo de comandos, estruturas e conceitos já apresentados em aula.

Referências

- **Veja material auxiliar em:** <https://github.com/jonathacosta-IA/PL>
 - *Slides* em: https://github.com/JonathaCosta-IA/PL/tree/main/A-PL_Slides
 - Códigos em: https://github.com/JonathaCosta-IA/PL/tree/main/B_PL_Codes
- **Referência basilares**
 - PUD da Disciplina de Lógica de Programação
 - DEITEL, P. J.; DEITEL, H. M. C: Como programar. 6. ed. São Paulo: Pearson, 2011. E-book. Disponível em: <https://plataforma.bvirtual.com.br>. Acesso em: 28 jun. 2025.