

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 múltiplas 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>
```

```
int main()
```

```
{ printf("Oi!");  
}
```

```
// biblioteca
```

```
// Função principal
```

```
// 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 *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():

```
1 int main() {  
2     saudacao();  
3     return 0;  
4 }
```

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():

```
1 int main() {  
2     int ano = obterAno();  
3     printf("%d\n", ano);  
4     return 0;  
5 }
```

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 dobraMatriz(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     dobraMatriz(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    }
```

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

Exercícios utilizando funções

Veja o slide 33 – Exercícios para explorar múltiplas soluções.

Organização

- 1 Blocos de programação - C
- 2 Modularização
 - Conceitos de modularização
 - Conceitos distintos de programação
- 3 Exercícios para explorar múltiplas 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

- ❶ Criar o arquivo “.c” (**ScriptDesejado.c**) contendo o *script* desejado;
- ❷ Criar um arquivo de biblioteca com a extensão ‘h’ (**biblioteca.h**) contendo o nome do arquivo “.c”;
- ❸ 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 particionar 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;  
}
```

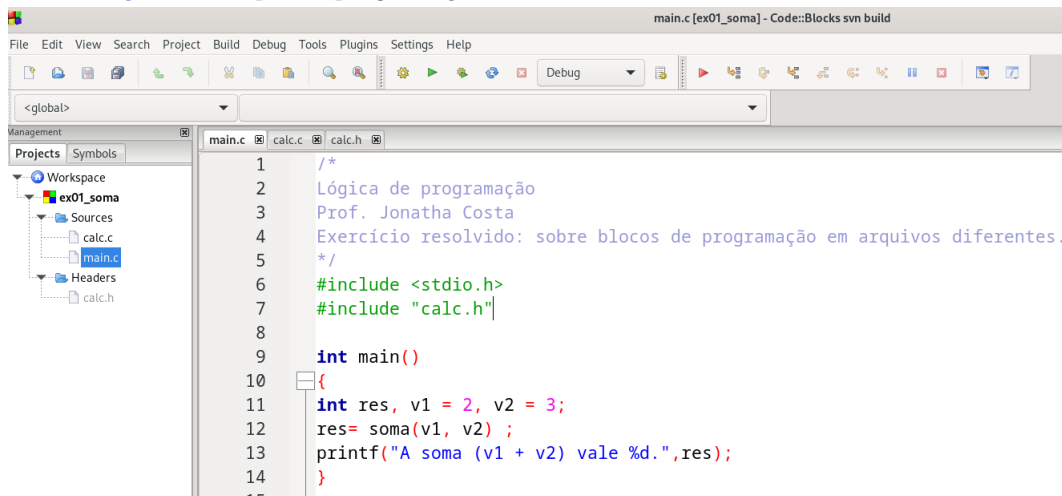
```
int subtracao(int v1,int v2)  
{ return v1-v2;  
}
```

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

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

Programa em módulos/blocos

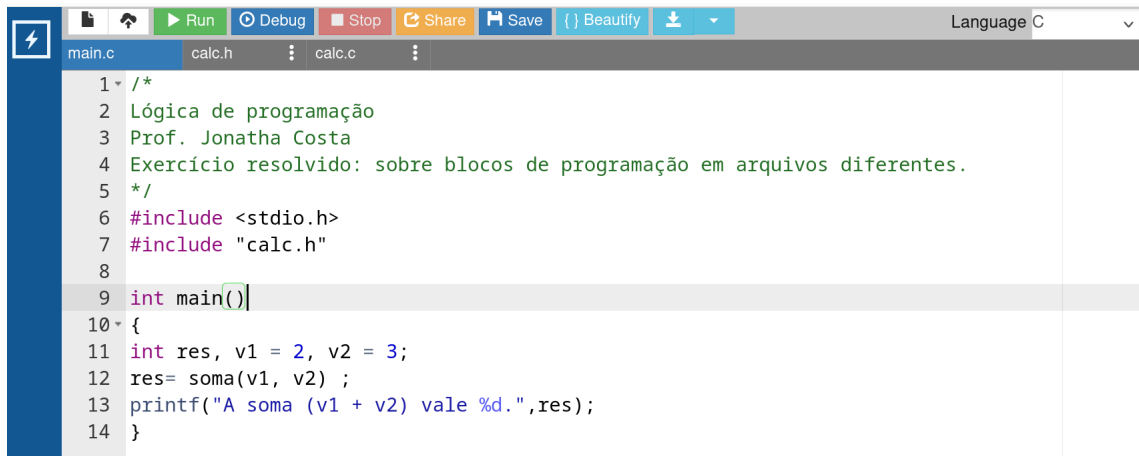
Figura: Exemplo de programação em blocos utilizando o Code Blocks® IDE



Fonte: AUTOR (2024)

Programa em módulos/blocos

Figura: Exemplo de programação em blocos utilizando o onlinedb[®] IDE



The screenshot shows the onlinedb IDE interface. The top toolbar includes icons for Run, Debug, Stop, Share, Save, Beautify, and a dropdown menu. The language is set to C. The file explorer shows three files: main.c, calc.h, and calc.c. The main.c file is open, displaying the following 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 numeros
5  */
6  #include<stdio.h>
7  int main()
8  {  int tam_vet=10;
9     int num[tam_vet];
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 }
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 numeros
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     {printf("Informe um número (%d / %d): ",i,tam_vet);
12        scanf("%d",&num[i]);}
13 }
14
15 int main()
16 { int tam_vet=10;
17   int num[tam_vet];
18   CarregarVetor(num,tam_vet);
19 }
20
```

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 numeros
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 }
15
```

(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      {printf("Informe um número (%d / %d): ",i,tam_vet)
7        scanf("%d",&num[i]);
8      }
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 numeros
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 múltiplas soluções

Exercícios para explorar múltiplas soluções

Exercícios de fixação para explorar com múltiplas 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 múltiplas soluções

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

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

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

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

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 múltiplas 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 múltiplas 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.