Engenharia da Computação

www.eComp.Poli.br

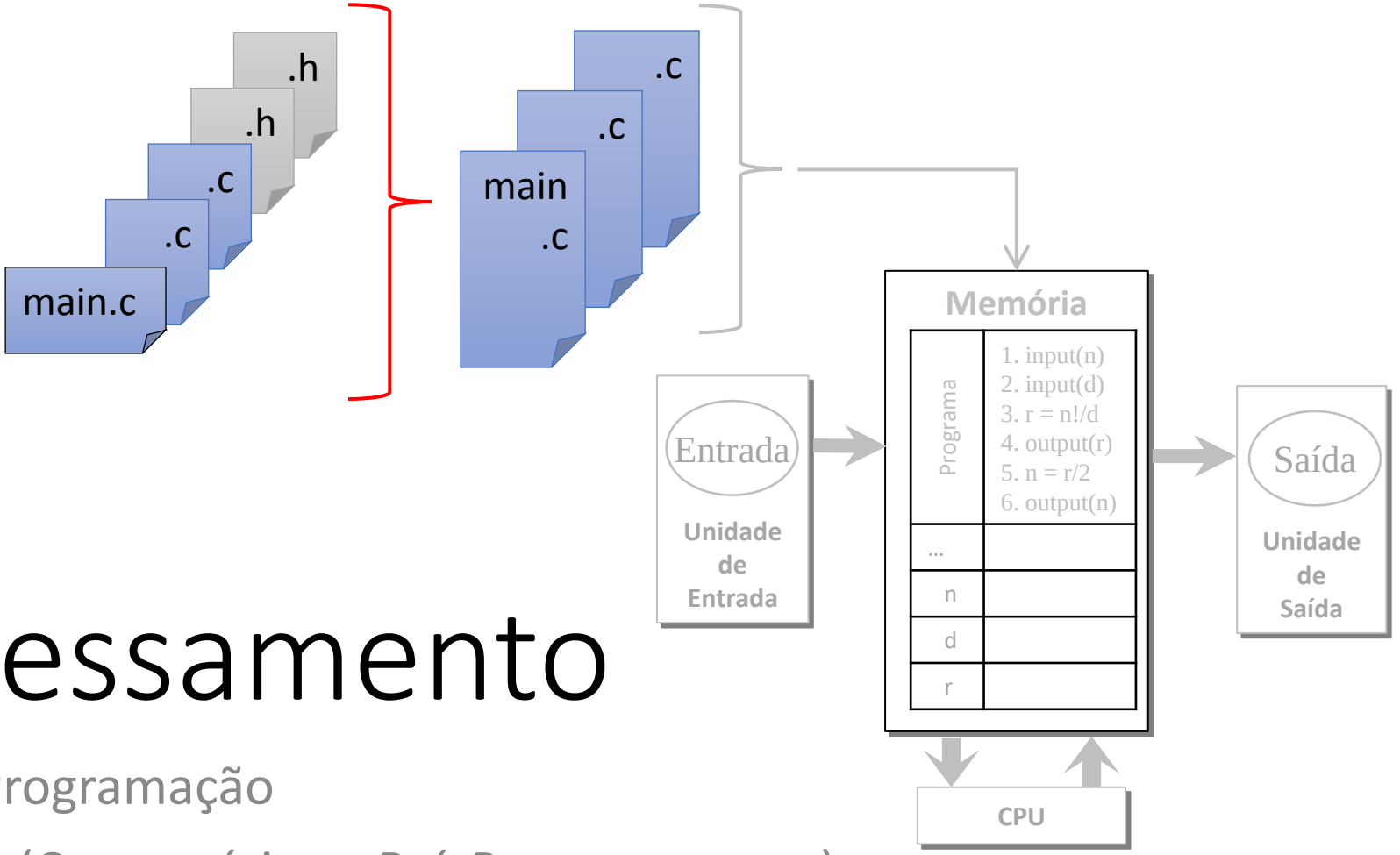
Pré-processamento

Disciplina: **DCExt Programação Imperativa**

Prof. Hemir Santiago

hcs2@poli.br

Material cedido pelo Prof. Joabe Jesus



Pré-processamento

Estilos de Programação

Metadados (Comentários e Pré-Processamento)

Header Files

Estilos de Programação

- Ao escrever seu programa, você pode colocar espaços, tabulação e pular linhas à vontade, pois o compilador ignora estes caracteres.
  - Em C não há um estilo obrigatório
- MAS sugerimos que se adotem boas práticas

Comentários

- É sempre bom adicionar comentários para explicar pontos de um programa
  - Ignorados pelo compilador
  - Importante para os programadores
  - Exemplo:

```
main()
{
   /* comentário sobre
  os próximos
  passos */
   ...
   //comentários de uma linha
}
```

Exemplos

```
main()
{
    /* Comentário sobre a constante PI
       descrevendo ...
     */
    const double PI = 3.1415;
    /*const*/ double sqrt2 = 1.41;
    double raio, area /*, volume */;
    raio = 4.0; // no fim da linha; i = i + 1;
    area = PI * raio * raio;
    // TODO volume = area * altura;
    // Comentários de linha única isolado
    ...
}
```

Diretivas de **pré-processamento**

- Permite que o programador modifique a compilação
- O pré-processador é um programa que examina e modifica o código-fonte antes da compilação
- As diretivas são os comandos utilizados pelo pré-processador
  - Estes comandos estarão disponíveis no código-fonte, mas não no código compilado

Diretivas de **pré-processamento**

`#include`

- A diretiva #include permite incluir (importar) funções definidas em outros arquivos (normalmente, ***Header Files***)

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

main()
{
    ...
}
```

Diretivas de **pré-processamento**

- Permite inserir um arquivo no código-fonte
- A diretiva *include* é substituída pelo conteúdo do arquivo
- Quando usamos <> para indicar o arquivo, este arquivo é procurado somente na pasta *include*
- Quando utilizamos “” para indicar o arquivo, este arquivo é procurado na pasta atual, e, se não for encontrado, é procurado na pasta *include*

Diretivas de **pré-processamento**

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

arquivo.c

```
#include "arquivo.c"

int main()
{
    soma(1, 2);
    return 0;
}
```

principal.c

Diretivas de **pré-processamento**

#define

- A diretiva #define permite definir ***macros*** (regras de substituição de texto)

```
#define UM 1
#define DOIS UM+UM

main()
{
     ...
     y = DOIS;
     x = y + UM;
     z = x + UM;
     ...
}
```

```
main()
{
     ...
     y = 1+1;
     x = y + 1;
     z = x + 1;
     ...
}
```

Diretivas de **pré-processamento**

- Permite definir constantes sem consumir memória durante a execução
- Não use o sinal de atribuição =

```
#define PI 3.14
int main()
{
    double raio = 1.0;
    double area = PI * raio * raio;
    ...
    return 0;
}
```

Diretivas de **pré-processamento**

- Permite definir trechos fixos de código

```
#define ERRO printf("Ocorreu um erro\n");
int main() {
    ERRO;
    ...
    return 0;
}
```

Diretivas de **pré-processamento**

- Permite definir trechos de código com parâmetros (macros)
- Não pode ter espaços no identificador. Ex.: SOMA (x,y)

```
#define SOMA(x,y) x + y
int main()
{
    int a = SOMA(1, 2);
    double b = SOMA(1.0, 2.0);
    ...
    return 0;
}
```

Diretivas de **pré-processamento**

- Recomenda-se usar parênteses em macros

```
#define SOMA(x,y) x + y
int main()
{
    printf("%d\n", 10 * SOMA(1,2));
    ...
    return 0;
}

// solução:  #define SOMA(x,y) (x + y)
```

Diretivas de pré-processamento

- Recomenda-se usar parênteses em macros

```
#define PRODUTO(x,y) (x * y)
int main()
{
    printf("%d\n", PRODUTO(2+3, 4));
    ...
    return 0;
}

// solução:  #define PRODUTO(x,y) ((x) * (y))
```

Diretivas de pré-processamento

- **#undef** remove a definição criada com #define

```
#define TAM_STRING 20

...

#undef TAM_STRING
#define TAM_STRING 100
```

Diretivas Condicionais

`#if`

- A diretiva #if permite enviar um trecho de código para o compilador se uma condição for **verdadeira**

`#else`

- A diretiva #else permite tratar o caso **falso** da condição definida na diretiva #if anterior

`#elif`

- Equivale a um #else #if

`#endif`

- Toda diretiva condicional deve ser delimitada pela diretiva #endif que finaliza o bloco de verificação

Diretivas Condicionais de Definição

`#ifdef`

- A diretiva #ifdef permite verificar se uma macro foi definida anteriormente (no arquivo atual ou em um arquivo previamente incluso)

`#ifndef`

- A diretiva #ifndef permite verificar se uma macro NÃO foi definida anteriormente

Diretivas Condicionais

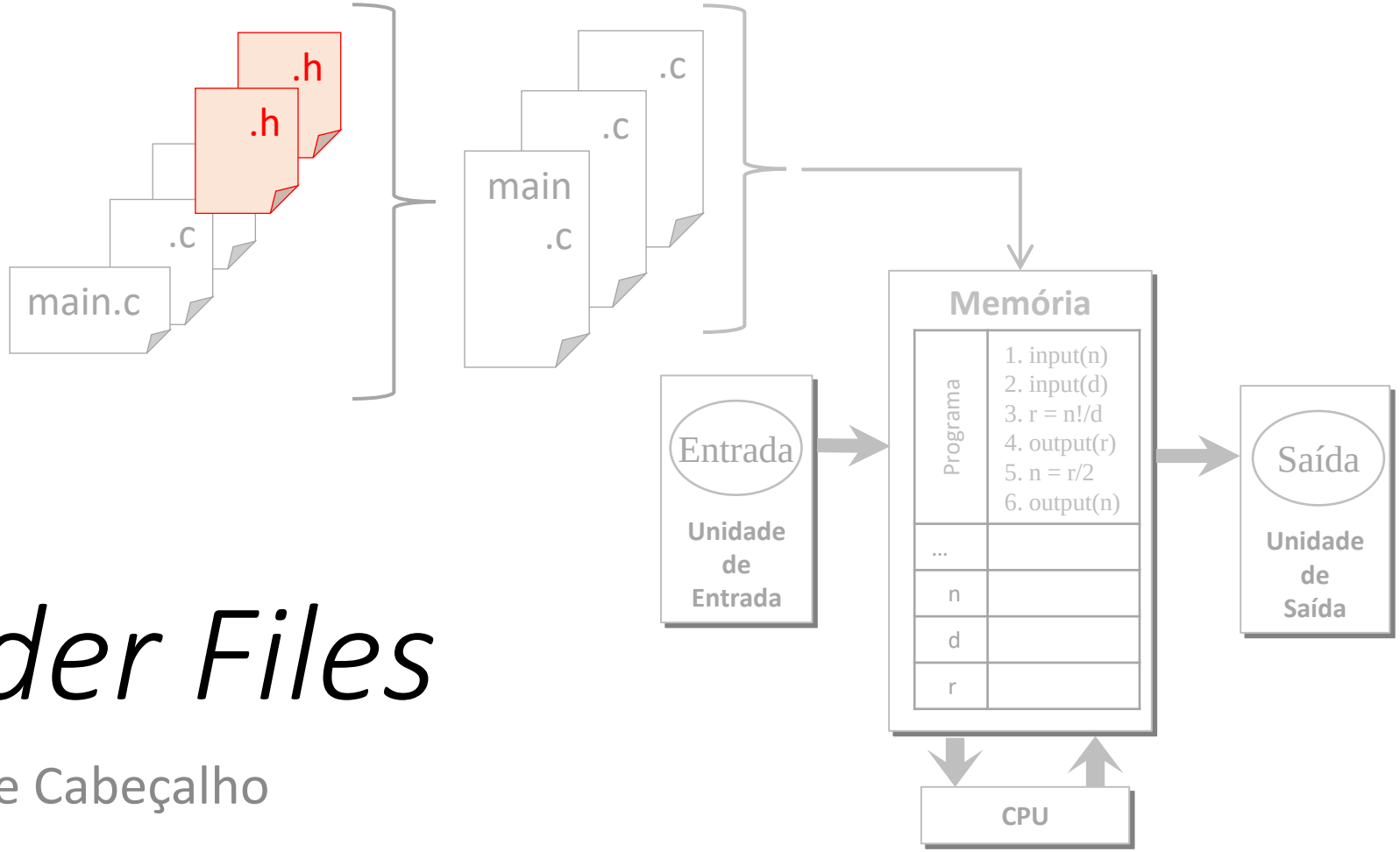
```
#define DEBUG 1

int main()
{
    ...
#if DEBUG == 1
    printf("Descricao detalhada: ...\n");
#elif DEBUG == 2
    printf("Descricao resumida: ...\n");
#else
    printf("Nenhuma descricao\n");
#endif
    ...
}
```

Diretivas Condicionais

```
#define DEBUG

int main()
{
    ...
#ifdef DEBUG
    printf("Descricao detalhada: ...\n");
#else
    printf("Nenhuma descricao\n");
#endif
    ...
}
```

Diretivas Condicionais

```
#include <stdio.h>
#define DEBUG 1

#if (DEBUG==1)
void main ()  {
   float soma = 0;
   int i, tamanho = 10;
   float vetor[] = {0,1,2,3,4,5,6,7,8,9};
   for (i=1; i<tamanho; i++)
            soma+=vetor[i];
   printf("A soma é = %.2f\n",soma);
} #else
void main ()  {
   float produto = 0;
   int i, tamanho = 10;
   float vetor[] = {0,1,2,3,4,5,6,7,8,9};

   for (i=1;i<tamanho;i++)
            produto*=vetor[i];
   printf("O produto é = %.2f\n", produto);
} #endif
```



Header Files

Arquivos de Cabeçalho

Header Files

- ***Header Files*** permitem estruturar/organizar Grandes Projetos
  - **Dividir para Conquistar**
    - Arquivos menores facilitam manutenção e compilação
  - Agrupamento
    - Cada arquivo fonte deve ter uma coleção de funções relacionadas e coerentes
    - Exemplo: imagens.h, audio.h, graficos.h, …

Header Files

- Com exceção do arquivo *main.c* outros códigos fonte devem ser separados em:

  <meuArquivo>.h

  - Possuirá tipos definido pelo programador (como *typedef's*, *enum's* e *struct's*)
  - Possuirá assinaturas/protótipos de funções (*function prototypes*) para cada função cujo corpo está definido em *<meuArquivo>.c*

  <meuArquivo>.c

  - Possuirá as definições (corpo) de funções
  - Pode possuir assinaturas de funções usadas **apenas** no próprio arquivo *<meuArquivo>.c*

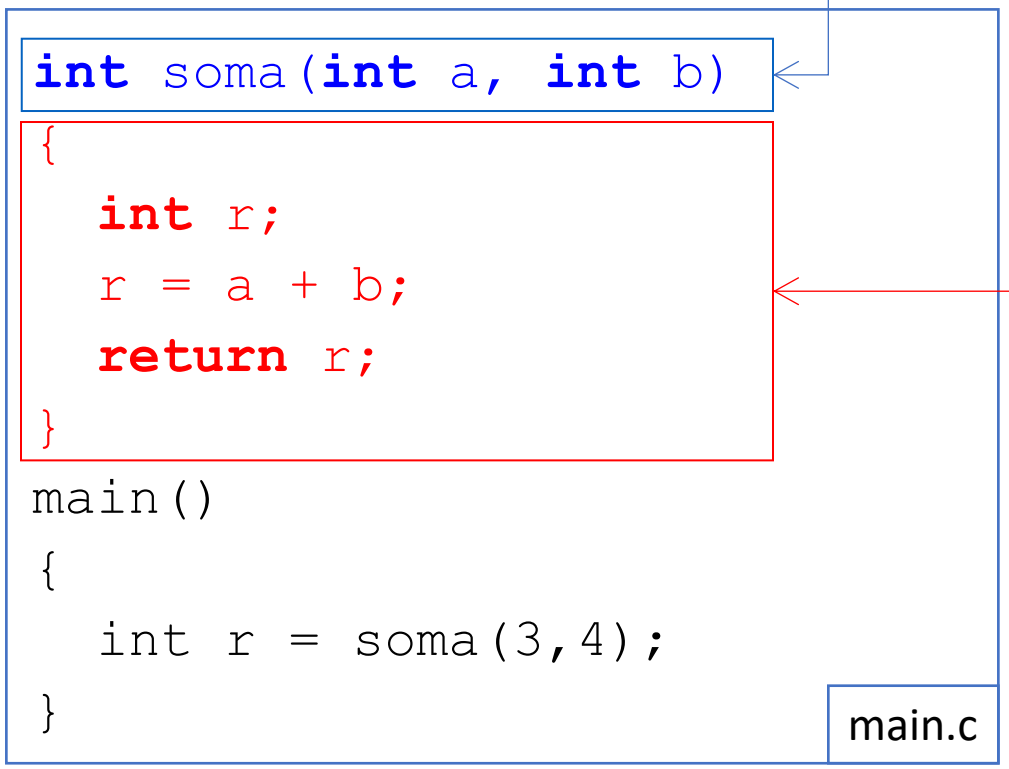
ATENÇÃO

1. Salve os ***header files*** (.h) na mesma pasta (***folder***) que estão seus arquivos fonte (.c)

- Use <…> para incluir um ***header file*** definido pela biblioteca padrão C
  - Exemplo: `#include <stdio.h>`
- Use "…" para incluir um ***header file*** próprio
  - Exemplo: `#include "minhasFuncoes.h"`

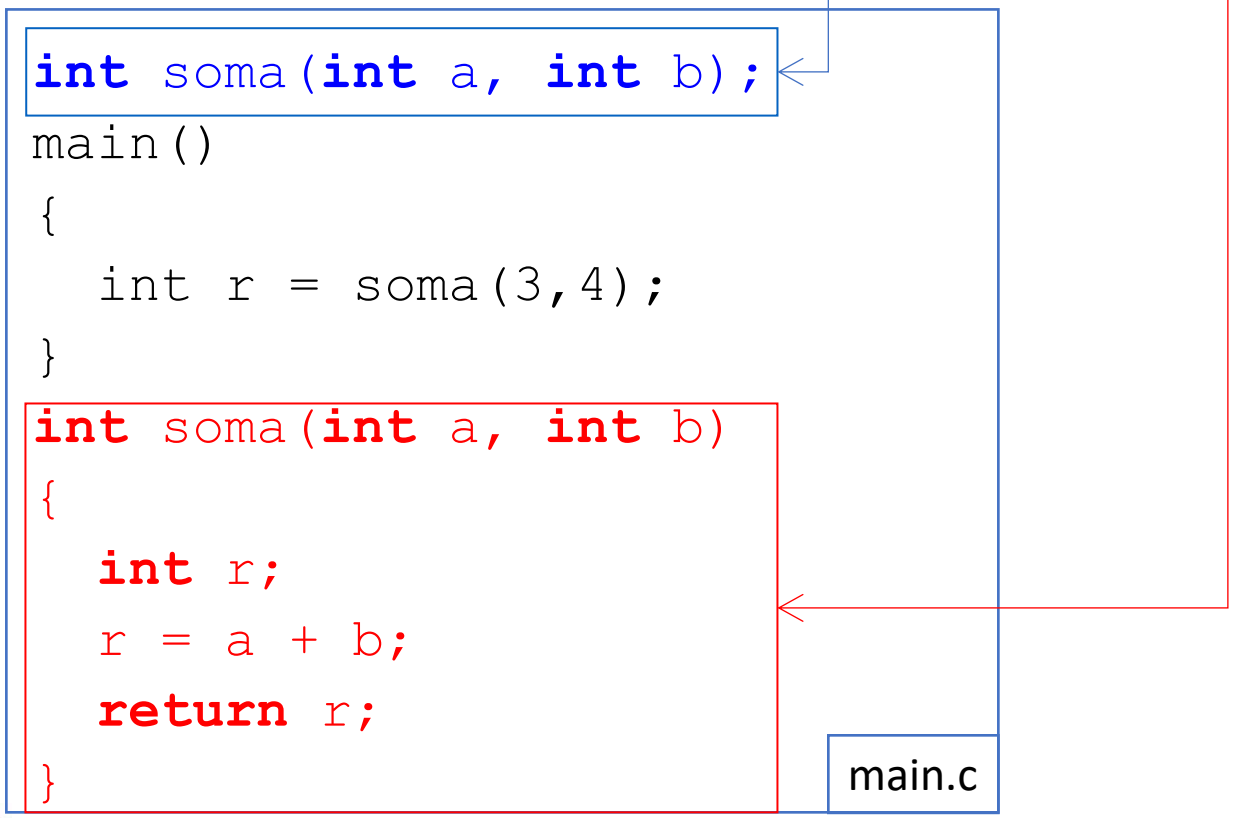
Dividir para Conquistar (1 de 3)

- Uma função em C possui assinatura e um corpo



Dividir para Conquistar (2 de 3)

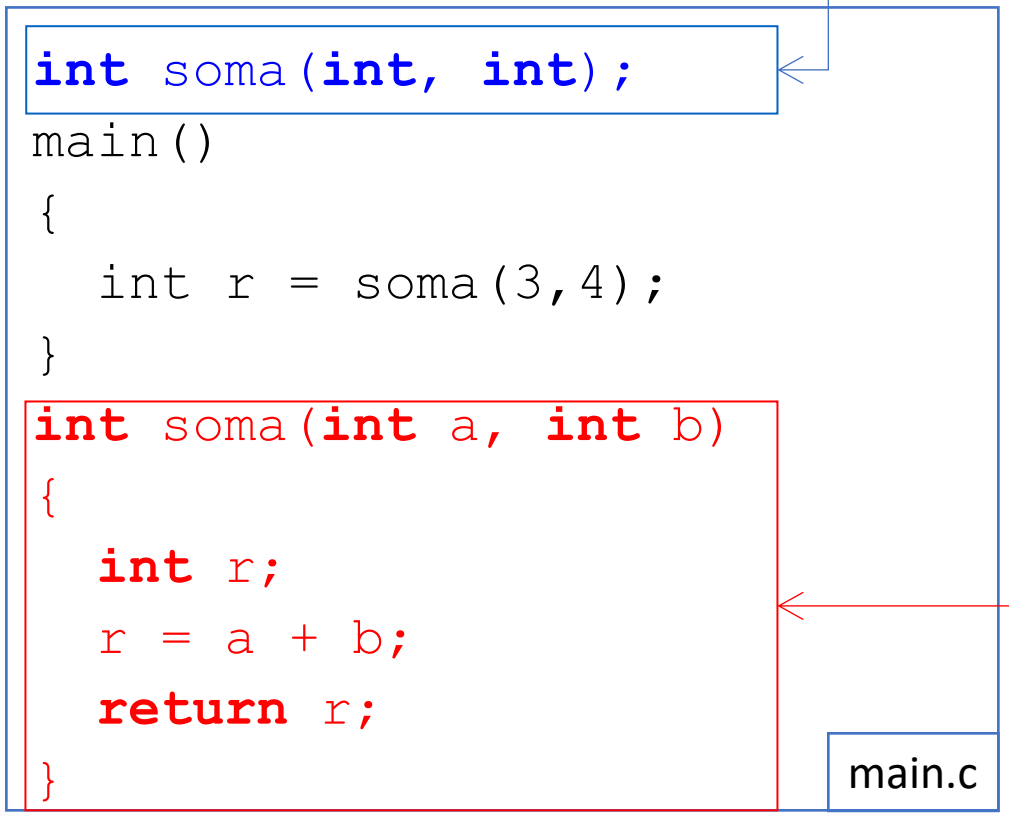
- Uma função em C possui assinatura e um corpo



OBSERVE A SEPARAÇÃO ENTRE A ASSINATURA E O CORPO. NOTE QUE O CORPO PRECISA REPETIR A ASSINATURA.

Dividir para Conquistar (2 de 3)

- Uma função em C possui assinatura e um corpo



OBSERVE QUE A ASSINATURA PODE OMITIR OS NOMES DOS PARÂMETROS.

Dividir para Conquistar (3 de 3)

- Uma função em C possui assinatura e um corpo

```c
int soma(int, int);
```

funcoes.h

```c
#include "funcoes.c"

main()
{
   int r = soma(3,4);
}
```

main.c

```c
#include "funcoes.h"

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

funcoes.c

NOTE QUE PODEMOS SEPARAR ASSINATURA NUM ARQUIVO DE CABEÇALHO E O CORPO NUM ARQUIVO .C
OBSERVE TAMBÉM O USO DE INCLUDE NOS DOIS ARQUIVOS .C

Exercício 01

Elabore um programa que solicita ao usuário digitar três números inteiros: *a*, *b* e *c*. Considere que esses números representam os coeficientes de uma equação do segundo grau, ou seja: $ax^2 + bx + c$. Após receber os três valores, o programa deve determinar as raízes da equação de segundo grau se existirem ou informar que não há raízes reais.

$$x = \frac{-b \pm \sqrt{\Delta}}{2a}; \quad \Delta = b^2 - 4ac$$

→ **discriminante positivo (Δ > 0)**: duas soluções para a equação;

→ **discriminante igual a zero (Δ = 0)**: as soluções da equação são repetidas;

→ **discriminante negativo (Δ < 0)**: não admite solução real.

Exercício 01

- Utilize a função *sqrt()* da biblioteca <math.h>
- O programa deve ser estruturado da seguinte forma:
  - 1 arquivo .c com a função *main()*
  - 1 arquivo .c com as funções que realizam os cálculos da equação
  - 1 arquivo .h com as assinaturas das funções

Exercício 02

Faça um programa que soma *n* números de duas formas:

- Se a diretiva estiver definida, os números são lidos direto do código;
- senão, os números são lidos do usuário com *scanf*.

Exercício 03

Elaborar programas para obter os seguintes somatórios (considere *n = 10)*:

$$\sum_{i=1}^{n} i \qquad \sum_{i=1}^{n} 2i \qquad \sum_{i=1}^{n} i^2 \qquad \sum_{i=1}^{n} i^3$$

(a) (b) (c) (d)

- O somatório a ser calculado será definido usando diretivas condicionais;
- Utilizar a estrutura: 1 arquivo principal .c + 1 arquivo .c com as funções + 1 arquivo de cabeçalho .h com as assinaturas das funções

	DATA	AULA
1	22/08/2024	Apresentação da disciplina | Introdução à Programação Imperativa
2	29/08/2024	Introdução à Linguagem de Programação C
3	05/09/2024	Conceitos Fundamentais
4	12/09/2024	Tipos de Dados Especiais em C
5	19/09/2024	Estruturas Condicionais e de Repetição
6	26/09/2024	Pré-processamento
7	03/10/2024	Registros/Estruturas de Dados
8	10/10/2024	Ponteiros
9	17/10/2024	1º Exercício Escolar

Plano de Aulas

	DATA	AULA
10	24/10/2024	Arquivos
11	31/10/2024	Acompanhamento de projetos
12	07/11/2024	Acompanhamento de projetos
13	14/11/2024	Acompanhamento de projetos
14	21/11/2024	Acompanhamento de projetos
15	28/11/2024	Apresentação parcial
16	05/12/2024	Apresentação de projetos
17	12/12/2024	Avaliação Final

Plano de Aulas