PROGRAMAÇÃO COMPUTACIONAL PARA ENGENHARIA

FUNÇÕES

Maurício Moreira Neto[1]

[1]**Universidade Federal do Ceará**
Departamento de Computação

30 de janeiro de 2020

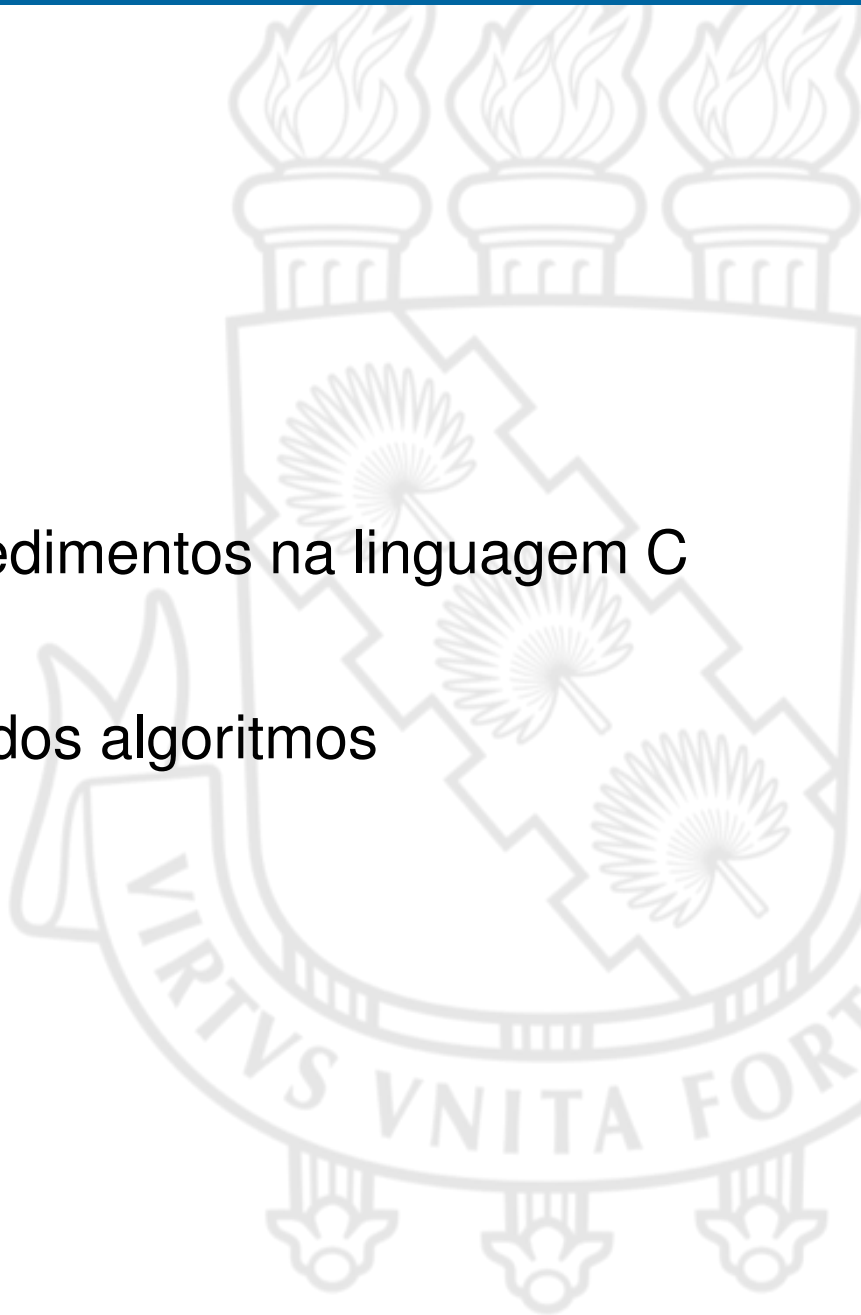
DEPARTAMENTO DE COMPUTAÇÃO

Sumário

DEPARTAMENTO DE COMPUTAÇÃO

Objetivos

- Aprender a criar funções e procedimentos na linguagem C
- Utilizar essas funções ao longo dos algoritmos



DEPARTAMENTO DE COMPUTAÇÃO

Função

- São blocos de códigos que podem ser nomeados e chamados de dentro de um programa
- Até o momento, já utilizamos diversas funções em nossos códigos
  - **printf()** – função que escreve na tela
  - **scanf()** – função que lê o teclado

Função

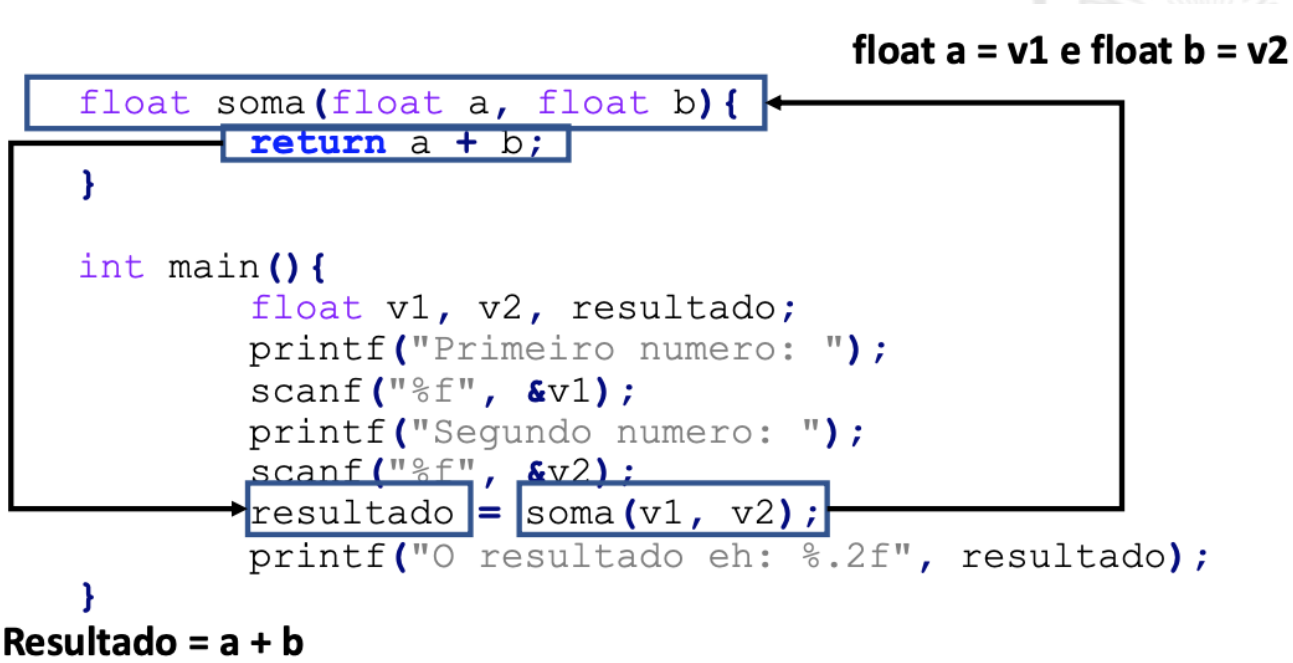
- Facilita a estruturação e reutilização do código

1. **Programação Estruturada:** programas grandes e complexos são construídos bloco a bloco (modularização)
2. **Reutilização:** utilizar funções evita a cópia desnecessária de trechos de códigos que realizam a mesma tarefa, fazendo diminuir o tamanho do programa e a ocorrência de erros

DEPARTAMENTO DE COMPUTAÇÃO

Função

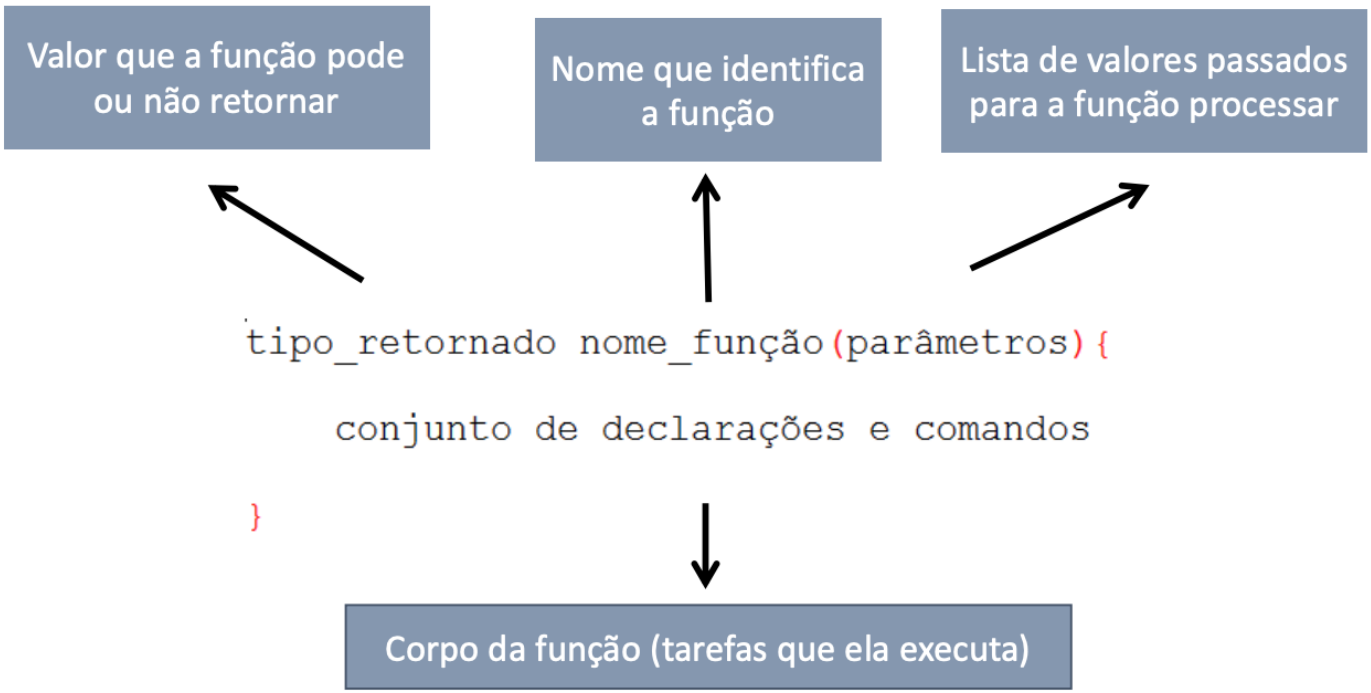
- Ao chamar uma função, o programa que a chamou é pausado até que a função termine a sua execução



DEPARTAMENTO DE COMPUTAÇÃO

Estrutura da função

- Forma geral de uma função



DEPARTAMENTO DE COMPUTAÇÃO

Corpo da função

- O corpo de uma função é a sua parte principal
  - Formada pelos comandos que função deve executar
  - Processa parâmetros (se houver), realizar outras tarefas e gerar saídas (se necessário)
  - Similar a claúsula **main()**

```
int main() {
// conjunto de comandos
return 0;
```

DEPARTAMENTO DE COMPUTAÇÃO

Corpo da função

- De modo geral, evita-se fazer operações de leitura e escrita dentro de uma função
  - Uma função é construída com o intuito de realizar uma tarefa específica e bem-definida
  - As operações de entrada e saída de dados (**scanf()** e **printf()**) devem ser feitas em quem chamou a função (por exemplo, na **main()**)
  - Isso assegura que a função construída possa ser utilizada nas mais diversas aplicações, garantindo a sua generalidade

DEPARTAMENTO DE COMPUTAÇÃO

Parâmetros da função

- A declaração de parâmetros é uma lista de variáveis juntamente com seus tipos:
  - `tipo1 nome1, tipo2 nome2, ···, tipoN nomeN`

- Pode-se definir quantos parâmetros achar necessários

```c
//Declaração CORRETA de parâmetros
int soma(int x, int y){
    return x + y;
}
//Declaração ERRADA de parâmetros
int soma(int x, y){
    return x + y;
}
```

DEPARTAMENTO DE COMPUTAÇÃO

Parâmetros da função

- Os parâmetros são informações que a função recebe do programa principal (ou de quem o chamou)
  - Não é preciso fazer a leitura das variáveis dos parâmetros dentro da função

```
int x = 2;
int y = 3;
int soma(int x, int y){
    return x + y;
}

int main(){
    int z = soma(2,3);

    return 0;
}
```

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

    scanf("%d",&x);
    scanf("%d",&y);

    return x + y;
}
```

Parâmetros da função

- Podemos criar uma função que não recebe nenhum parâmetro de entrada
- Isso pode ser feito de duas formas:
  - Deixar a lista de parâmetros vazia
  - Colocar void entre os parênteses

```c
void imprime(){
    printf("Teste\n");
}

void imprime(void){
    printf("Teste\n");
}
```

DEPARTAMENTO DE COMPUTAÇÃO

Retorno da função

- Uma função pode ou não retornar um valor
  - Se ela retornar um valor, alguém deverá receber este valor
  - Uma função que retorna nada é definida colocando-se o tipo **void** como valor retornado

- Podemos retornar qualquer valor válido em C
  - Tipos pré-definidos: **int**, **char**, **float** e **double**
  - Tipos definidos pelo usuário: **struct**

DEPARTAMENTO DE COMPUTAÇÃO

Comando return

- O valor retornado por uma função pode ser dado pelo comando **return**
- Forma geral:
  - **return** `expressão ou valor;`
  - **return;**
    - Usada para terminar uma função que não retorna um determinado valor
- **Importante**: lembrar que o valor de retorno fornecida tem que ser compatível com o tipo de retorno declarado para a função

DEPARTAMENTO DE COMPUTAÇÃO

Comando return

Função com o retorno de valor

```c
int soma(int x, int y){
    return x + y;
}

int main(){
    int z = soma(2,3);

    return 0;
}
```

Função sem o retorno de valor

```c
void imprime(){
    printf("Teste\n");
}

int main(){
    imprime();

    return 0;
}
```

DEPARTAMENTO DE COMPUTAÇÃO

Comando return

- Uma função pode ter mais de uma declaração **return**
  - Quando o comando **return** é executado, a função termina imediatamente
  - Todos os comandos restantes são **ignoradas**

```
int maior(int x, int y){
    if(x > y)
        return x;
    else
        return y;
        printf("Esse texto nao sera impresso\n");
}
```

DEPARTAMENTO DE COMPUTAÇÃO

Declaração da função

- Funções devem ser declaradas antes de serem utilizadas, ou seja, antes da cláusula **main**
  - Uma função criada pelo programador pode ser utilizada qualquer outra função, inclusive as que foram criadas

```
int quadrado(int a){
    return a*a;
}

int main(){
    int n1,n2;
    printf("Entre com um numero: ");
    scanf("%d", &n1);

    n2 = quadrado(n1);

    printf("O seu quadrado vale: %d\n", n2);
    return 0;
}
```

DEPARTAMENTO DE COMPUTAÇÃO

Declaração da função

- Podemos definir apenas o protótipo da funcao antes da cláusula **main**
  - O protótipo apenas indica a existência da função
  - Assim, ela pode ser declarada após a cláusula **main()**

```
tipo_retornado nome_funcao(parametros);
```

Declaração da função

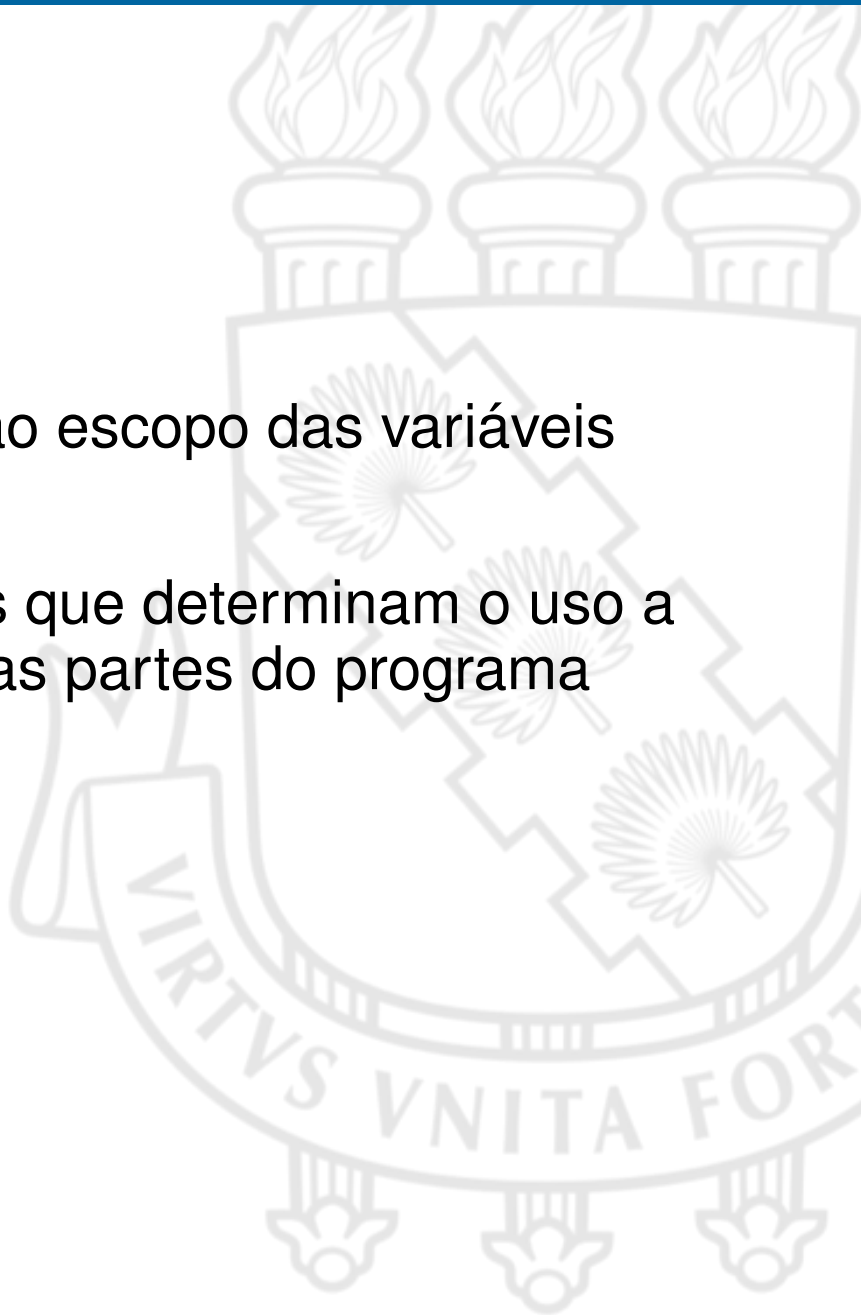
- Exemplo de protótipo

```c
int quadrado(int a);

int main(){
    int n1,n2;
    printf("Entre com um numero: ");
    scanf("%d", &n1);

    n2 = quadrado(n1);

    printf("O seu quadrado vale: %d\n", n2);
    return 0;
}

int quadrado(int a){
    return a*a;
}
```

DEPARTAMENTO DE COMPUTAÇÃO

Escopo

- Função também estão sujeitos ao escopo das variáveis
- O escopo é o conjunto de regras que determinam o uso a validade de variáveis nas diversas partes do programa
  - Variáveis Locais
  - Variáveis Globais
  - Parâmetros Formais



DEPARTAMENTO DE COMPUTAÇÃO

Escopo

- Variáveis locais são aqueles que só tem validade dentro do bloco no qual são declaradas
  - Um bloco começa quando abrimos uma chave e termina quando fechamos a chave
  - Exemplo: variáveis declaradas dentro da função

```
int fatorial (int n){
    if (n == 0)
        return 1;
    else{
        int i;
        int f = 1;
        for(i = 1; i <= n; i++)
            f = f * i;
        return f;
    }
}
```

DEPARTAMENTO DE COMPUTAÇÃO

Escopo

- Parâmetros formais são declaradas como sendo as entradas de uma função
  - Os parâmetros formais são variáveis locais da função
    - **Exemplo**: 'x' é um parâmetro forma

```
float quadrado(float x);
```

Escopo

- Variáveis globais são delcaradas fora de todas as funções do programa
- Estas variáveis são conhecidas e podem ser alteradas por todas as funções do programa
  - Quando uma função tem uma variável local com o mesmo nome de um variável global, a função dará preferência a variável local
- **IMPORTANTE:** é sempre bom evitar variáveis globais

DEPARTAMENTO DE COMPUTAÇÃO

Passagem de parâmetros

- Na linguagem C, os parâmetros de uma função são sempre passados por **valor**, ou seja, uma cópia do valor do parâmetro é feita e passada para a função
- Mesmo que esse valor mude dentro da função, nada acontece com o valor fora da função
- Mas podemos ter dois tipos de passagens de parâmetros
  - **Por valor**
  - **Por referência**

DEPARTAMENTO DE COMPUTAÇÃO

Passagem por valor

```
                                        int n = x;
void incrementa(int n){
    n = n + 1;

    printf("Dentro da funcao: x = %d\n",n);
}

int main(){
    int x = 5;
    printf("Antes da funcao: x = %d\n",x);

    incrementa(x);

    printf("Depois da funcao: x = %d\n",x);
    return 0;
}
```

Saída:

```
Antes da funcao: x = 5
Dentro da funcao: x = 6
Depois da funcao: x = 5
```

DEPARTAMENTO DE COMPUTAÇÃO

Passagem por referência

- Quando se quer que o valor da variável mude dentro da função, usa-se passagem de parâmetros por **referência**
- Neste tipo de chamada, não se passa a função o valor da variável, mas a sua **referência** (seu endereço na memória)

DEPARTAMENTO DE COMPUTAÇÃO

Passagem por referência

- Utilizando o endereço da variável, qualquer alteração que a variável sofra dentro da função será refletida fora da função
- Exemplo: função **scanf()**

Passagem por referência

- Exemplo: função **scanf()**
  - Sempre que desejamos ler algo do teclado, passamos para a função **scanf()** o nome da variável onde o dado será armazenado

- Essa variável tem seu valor modificado dentro da função **scanf()**, e seu valor pode ser acessado no programa principal

```c
int main(){
    int x = 5;
    printf("Antes do scanf: x = %d\n",x);
    printf("Digite um numero: ");
    scanf("%d",&x);
    printf("Depois do scanf: x = %d\n",x);

    return 0;
}
```

DEPARTAMENTO DE COMPUTAÇÃO

Passagem por referência

- Para passar um parâmetro por referência, coloca-se um asterisco '*' na frente do nome do parâmetro na declaração da função:

```c
//passagem de parâmetro por valor
void incrementa(int n);

//passagem de parâmetro por referência
void incrementa(int *n);
```

- Ao se chamar a função, é necessário agora utilizar o operador '&', assim como feito pela função **scanf()**

```c
//passagem de parâmetro por valor
int x = 10;
incrementa(x);

//passagem de parâmetro por referência
int x = 10;
incrementa(&x);
```

DEPARTAMENTO DE COMPUTAÇÃO

Passagem por referência

- No corpo da função, é necessário colocar um asterisco '*' sempre que se deseja acessar o conteúdo do parâmetro passado por referência

```
//passagem de parâmetro por valor
void incrementa(int n){
    n = n + 1;
}
//passagem de parâmetro por referência
void incrementa(int *n){
    *n = *n + 1;
}
```

DEPARTAMENTO DE COMPUTAÇÃO

Passagem por referência

```
int *n = &x;
void incrementa(int *n){
    *n = *n + 1;

    printf("Dentro da funcao: x = %d\n",n);
}

int main(){
    int x = 5;
    printf("Antes da funcao: x = %d\n",x);

    incrementa(&x);

    printf("Depois da funcao: x = %d\n",x);
    return 0;
}
```

Saída:

```
Antes da funcao: x = 5
Dentro da funcao: x = 6
Depois da funcao: x = 6
```

DEPARTAMENTO DE COMPUTAÇÃO

Exercício 1

1. Crie uma função que troque o valor de dois números inteiros passados por referência
   - **Dica: lembre-se da variável de auxílio**

2. Crie uma função que faça a soma de dois valores inteiros passados por referência.

DEPARTAMENTO DE COMPUTAÇÃO

Resolução

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

void trocar(int *a, int *b){
    int aux;
    aux = *a;
    *a = *b;
    *b = aux;
    printf("Valor de A: %d e valor de B: %d", *a, *b);
}

int somar (int *a, int *b){
    return *a + *b;
}

int main()
{
    int valor_1, valor_2, res;
    printf("Valor de A: ");
    scanf("%d", &valor_1);
    printf("Valor de B: ");
    scanf("%d", &valor_2);
    trocar(&valor_1, &valor_2);
    res = somar(&valor_1, &valor_2);
    printf("\nO valor da soma eh: %d", res);
    return 0;
}
```

DEPARTAMENTO DE COMPUTAÇÃO

Array como Parâmetros

- Para utilizar arrays como parâmetros de funções alguns cuidados simples são necessários
- Arrays são sempre passados por referência para uma função:
  - A passagem de **array por referência** evita cópia desnecessária de grandes quantidades de dados para outras áreas de memória durante a chamada da função, o que afetaria o desempenho do programa

DEPARTAMENTO DE COMPUTAÇÃO

Array como Parâmetros

- É necessário declarar um segundo parâmetro (em geral uma variável inteira) para passar a função o tamanho do array separadamente
  - Quando passamos um array por parâmetro, independente do seu tipo, o que é de fato passado é o endereço do primeiro elemento do array

Array como Parâmetros

- Na passagem de um array como parâmetro de uma função podemos declarar a função de diferentes maneiras, todas equivalentes:

```
void imprime(int *m, int n);
void imprime(int m[], int n);
void imprime(int m[5], int n);
```

DEPARTAMENTO DE COMPUTAÇÃO

Array como Parâmetros

- **Exemplo:**
  - Função que imprime um array

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

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

    return 0;
}
```

Memória		
posição	variável	conteúdo
119		
120		
121	int vet[5]	123
122		
123	vet[0]	1
124	vet[1]	2
125	vet[2]	3
126	vet[3]	4
127	vet[4]	5
128		

DEPARTAMENTO DE COMPUTAÇÃO

Array como Parâmetros

- Foi visto que para arrays, não é necessário especificar o número de elementos para a função

```
void imprime(int *m, int n);
void imprime(int m[], int n);
```

- No entanto, para arrays com mais de uma dimensão, é necessário especificar o tamanho de todas as dimensões, exceto a primeiro

```
void imprime(int m, int n);
```

DEPARTAMENTO DE COMPUTAÇÃO

Array como Parâmetros

- Na passagem de um array para uma função, o compilador precisar saber o tamanho de cada elemento, não o número de elementos
- Uma matriz pode ser interpretada como um array de arrays
  - **int m[4][5]** – array de 4 elementos onde cada elemento é um array de 5 posições inteiros

DEPARTAMENTO DE COMPUTAÇÃO

Array como Parâmetros

- Logo, o compilador precisa saber o tamanho de cada elemento do array

```
int m[4][5];
void imprime(int m[][5], int n);
```

- Na notação acima, informamos ao compilador que estamos passando um array, onde cada elemento dele é outro array de 5 posição inteiras

DEPARTAMENTO DE COMPUTAÇÃO

Array como Parâmetros

- Isso é necessário para que o programa saiba que o array possui mais de uma dimensão e mantenha a notação de um conjunto de colchetes por dimensão

- As notações abaixo funcionam para arrays com mais de uma dimensão. Mas o array é tratado como se tivesse apenas uma dimensão dentro da função

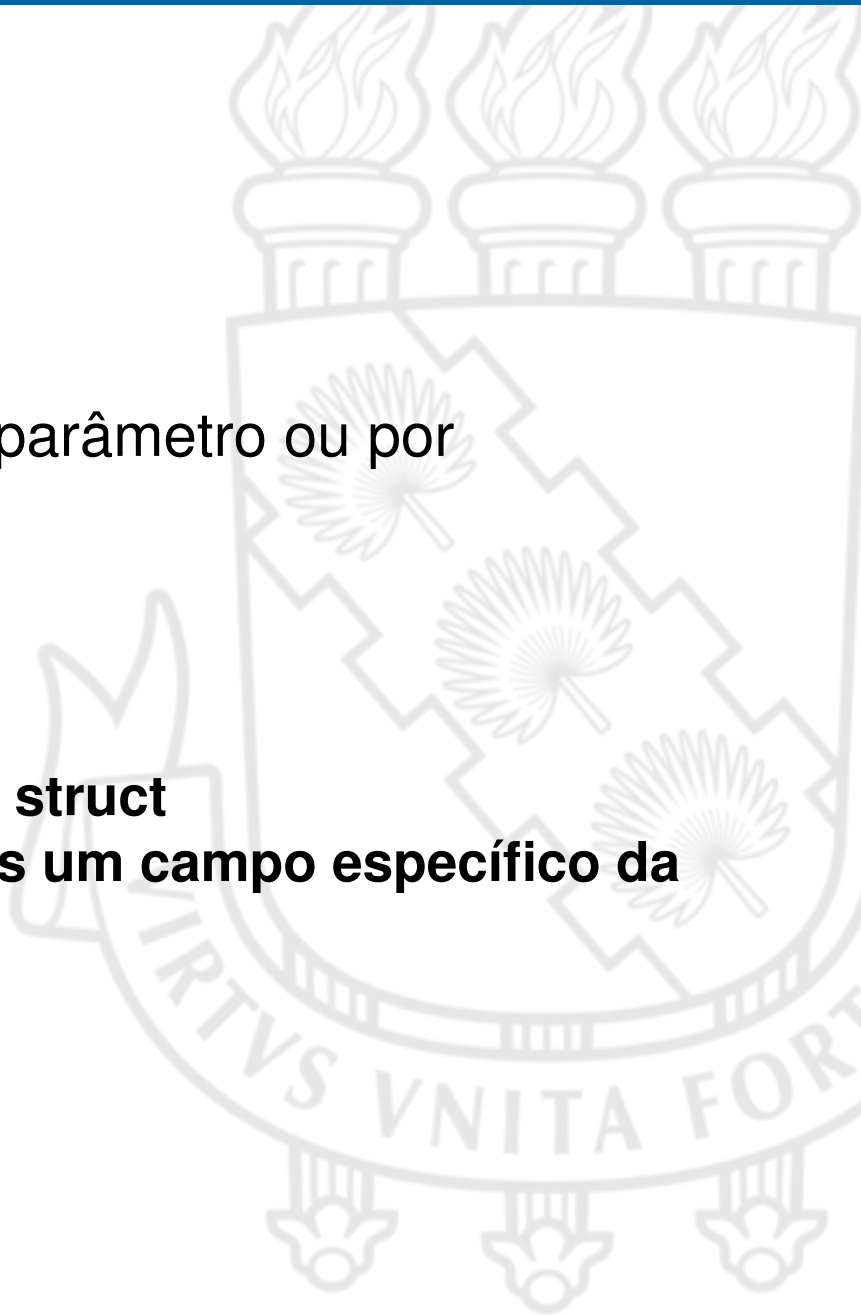
```
void imprime(int *m, int n);
void imprime(int m[], int n);
```

DEPARTAMENTO DE COMPUTAÇÃO

Struct como Parâmetros

- Pode-se passar uma **struct** por parâmetro ou por referência

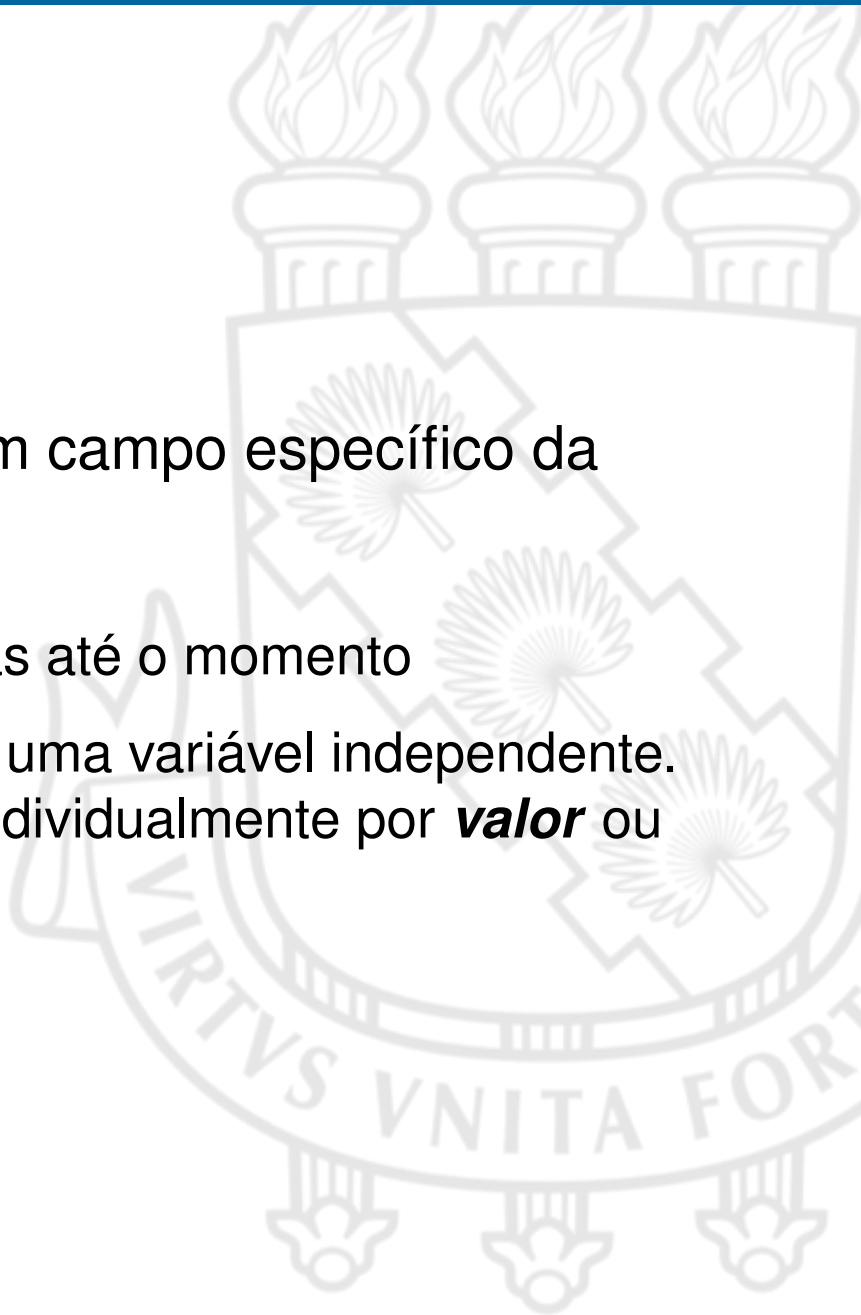
- Temos duas possibilidades:
  1. **Passar por parâmetro toda a struct**
  2. **Passar por parâmetro apenas um campo específico da struct**



DEPARTAMENTO DE COMPUTAÇÃO

Struct como Parâmetros

- Passar por parâmetro apenas um campo específico da **struct**
  - Valem as mesmas regras vistas até o momento
  - Cada campo da struct é como uma variável independente. Pode, portanto, ser passada individualmente por ***valor*** ou por ***referência***



DEPARTAMENTO DE COMPUTAÇÃO

Struct como Parâmetros

- Passar por parâmetro toda a **struct**
- Passagem por valor
  - Valem as mesmas regras vistas anteriormente
  - A **struct** é tratada como uma variável qualquer e seu valor é copiado para dentro da função
- Passagem por referência
  - Valem as regras de uso do asterisco '*' e operador de endereço '&'
  - Deve-se acessar o conteúdo da struct para somente depois acessar os seus campos e modificá-los
  - Uma alternativa é usar o **operador seta '–>'**

DEPARTAMENTO DE COMPUTAÇÃO

FStruct como Parâmetros

Usando '*'

```
struct ponto {
    int x, y;
};

void atribui(struct ponto *p){
    (*p).x = 10;
    (*p).y = 20;
}

struct ponto p1;

atribui(&p1);
```

Usando '->'

```
struct ponto {
    int x, y;
};

void atribui(struct ponto *p){
    p->x = 10;
    p->y = 20;
}

struct ponto p1;

atribui(&p1);
```

DEPARTAMENTO DE COMPUTAÇÃO

Recursão

- Na linguagem C, uma função pode chamar outra função
  - A função **main()** pode chamar qualquer outra função, seja da biblioteca da linguagem (como a **printf()**) ou definida pelo programador (função **imprime()**)

- Uma função também pode chamar a si própria
  - Chama-se esse tipo de função de **função recursiva**

DEPARTAMENTO DE COMPUTAÇÃO

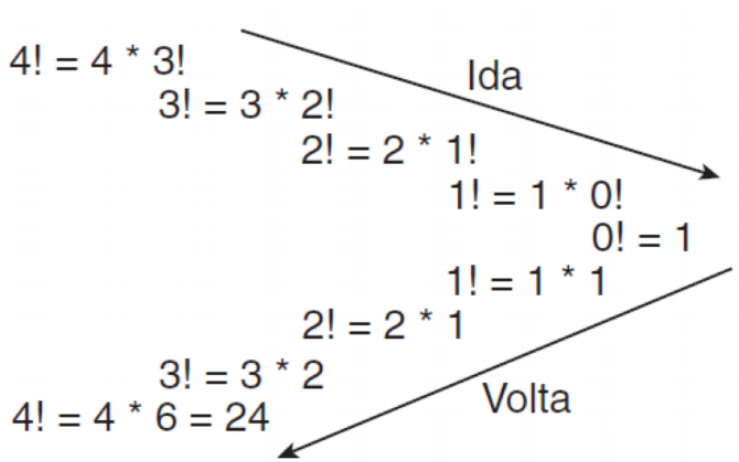
Recursão

- A recursão também é chamado de definição circular. Ocorre quando algo é definido em termos de si mesmo
- Um exemplo clássico de função que usa recursão é o cálculo do fatorial de um número:
  1. $3! = 3 * 2!$
  2. $4! = 4 * 3!$
  3. $n! = n * (n - 1)!$

DEPARTAMENTO DE COMPUTAÇÃO

Recursão

$$0! = 1$$
$$1! = 1 * 0!$$
$$2! = 2 * 1!$$
$$3! = 3 * 2!$$
$$4! = 4 * 3!$$



$n! = n * (n - 1)!$: fórmula geral

$0! = 1$: caso-base

DEPARTAMENTO DE COMPUTAÇÃO

Recursão

Com Recursão

```c
int fatorial(int n){
    if (n == 0)
        return 1;
    else
        return n * fatorial(n-1);
}
```

Sem Recursão

```c
int fatorial (int n){
    if (n == 0)
        return 1;
    else{
        int i;
        int f = 1;
        for(i = 1; i <= n; i++)
            f = f * i;
        return f;
    }
}
```

DEPARTAMENTO DE COMPUTAÇÃO

Recursão

- Em geral, formulações recursivas de algoritmos são frequentemente consideradas “mais enxutas” ou “mais elegantes” do que formulações iterativas

- Porém, algoritmos recursivos tendem a necessitar de mais espaço do que algoritmos iterativos

Recursão

- Todo cuidado é pouco ao se fazer funções recursivas
  - **Critério de parada**: determina quando a função deverá parar de chamar a si mesma
  - O parâmetro da chamada recursiva deve ser sempre modificado, de forma que a recursão chegue a um término

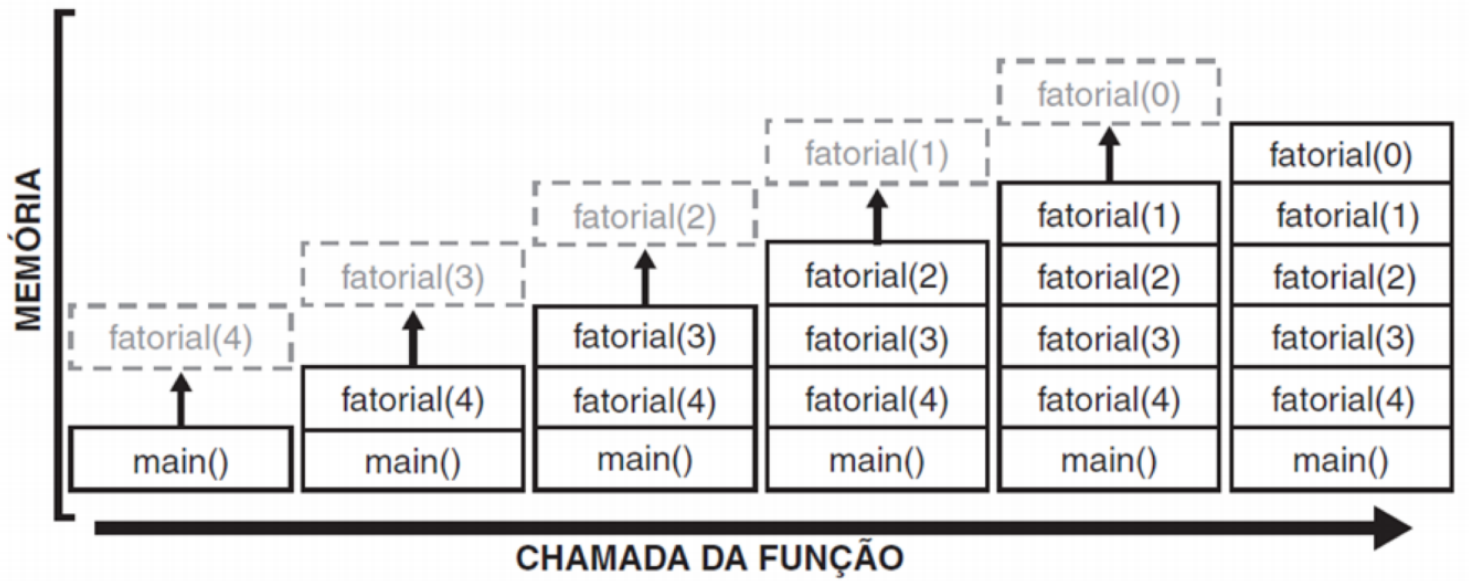
```
int fatorial (int n){
    if (n == 0)//critério de parada
        return 1;
    else /*parâmetro de fatorial sempre muda*/
        return n*fatorial(n-1);
}
```

DEPARTAMENTO DE COMPUTAÇÃO

Recursão

- O que acontece na chamada da função fatorial com um valor como n = 4

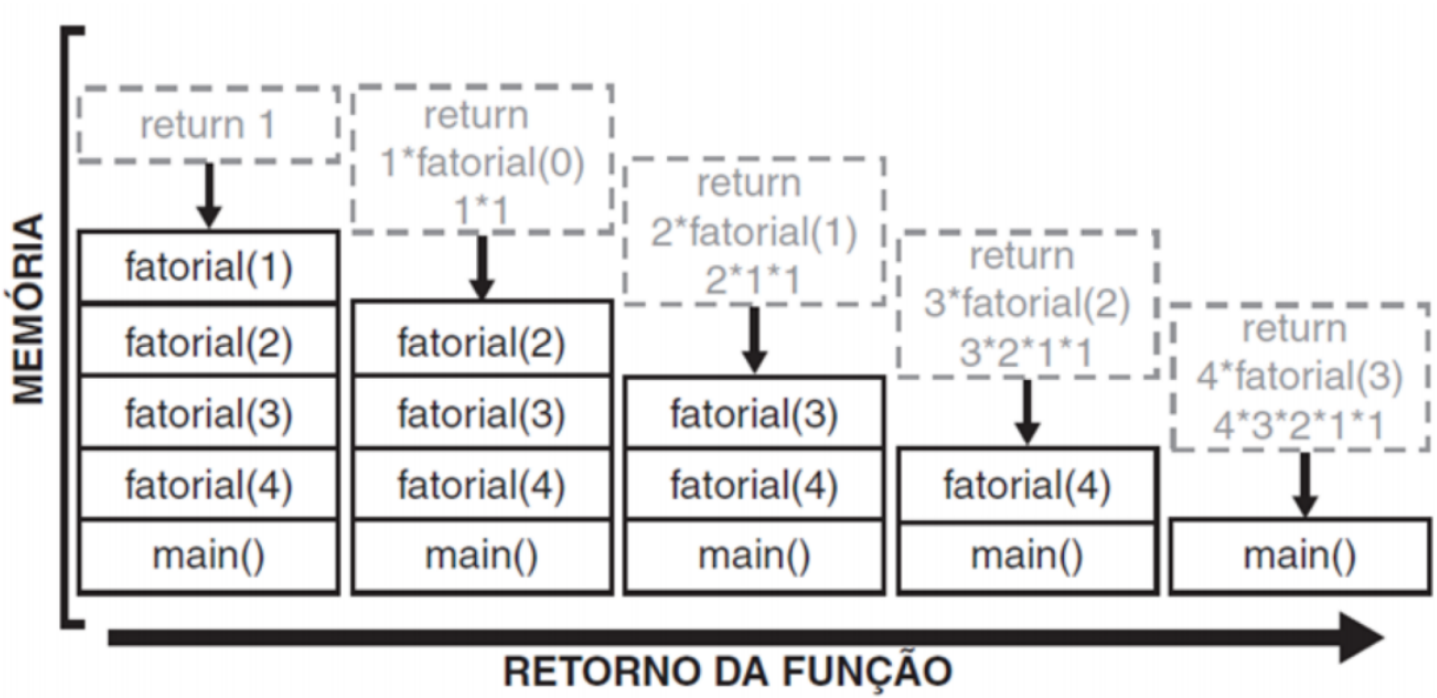
```
int x = fatorial(4);
```



DEPARTAMENTO DE COMPUTAÇÃO

Recursão

- Uma vez que chegamos ao caso-base, é hora de fazer o caminho de volta da recursão



DEPARTAMENTO DE COMPUTAÇÃO

Problema de Fibonacci

- Essa sequencia é recursiva:
  - 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, $\cdots$
- A sequência de Fibonacci é definida como uma função recursiva utilizando a fórmula a seguir

$$F(n) = \begin{cases} 0, & \text{se n = 0} \\ 1, & \text{se n = 1} \\ F(n-1) + F(n-2), & \text{outros casos} \end{cases}$$

- Sua solução recursiva é mais interessante!

DEPARTAMENTO DE COMPUTAÇÃO

Problema de Fibonacci

Com Recursão

```
int fibo(int n){
    int i, t, c, a = 0, b = 1;
    for(i = 0; i < n; i++){
        c = a + b;
        a = b;
        b = c;
    }
    return a;
}
```

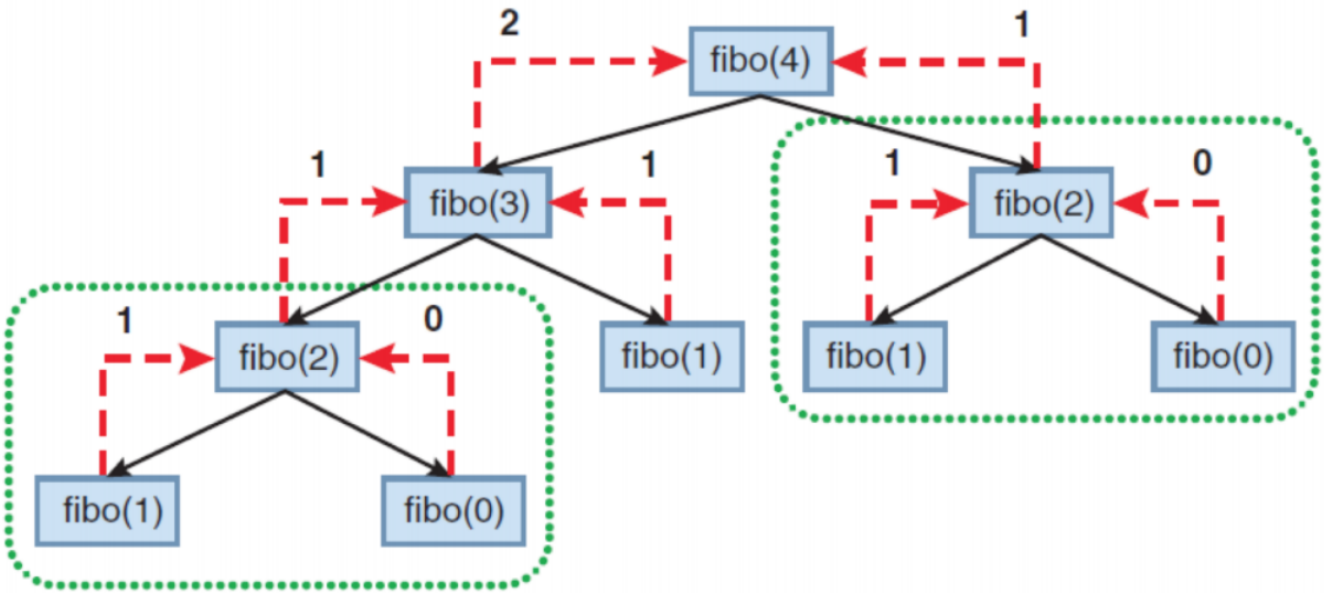
Sem Recursão

```
int fiboR(int n){
    if (n == 0 || n == 1)
        return n;
    else
        return fiboR(n-1) + fiboR(n-2);
}
```

DEPARTAMENTO DE COMPUTAÇÃO

Problema de Fibonacci

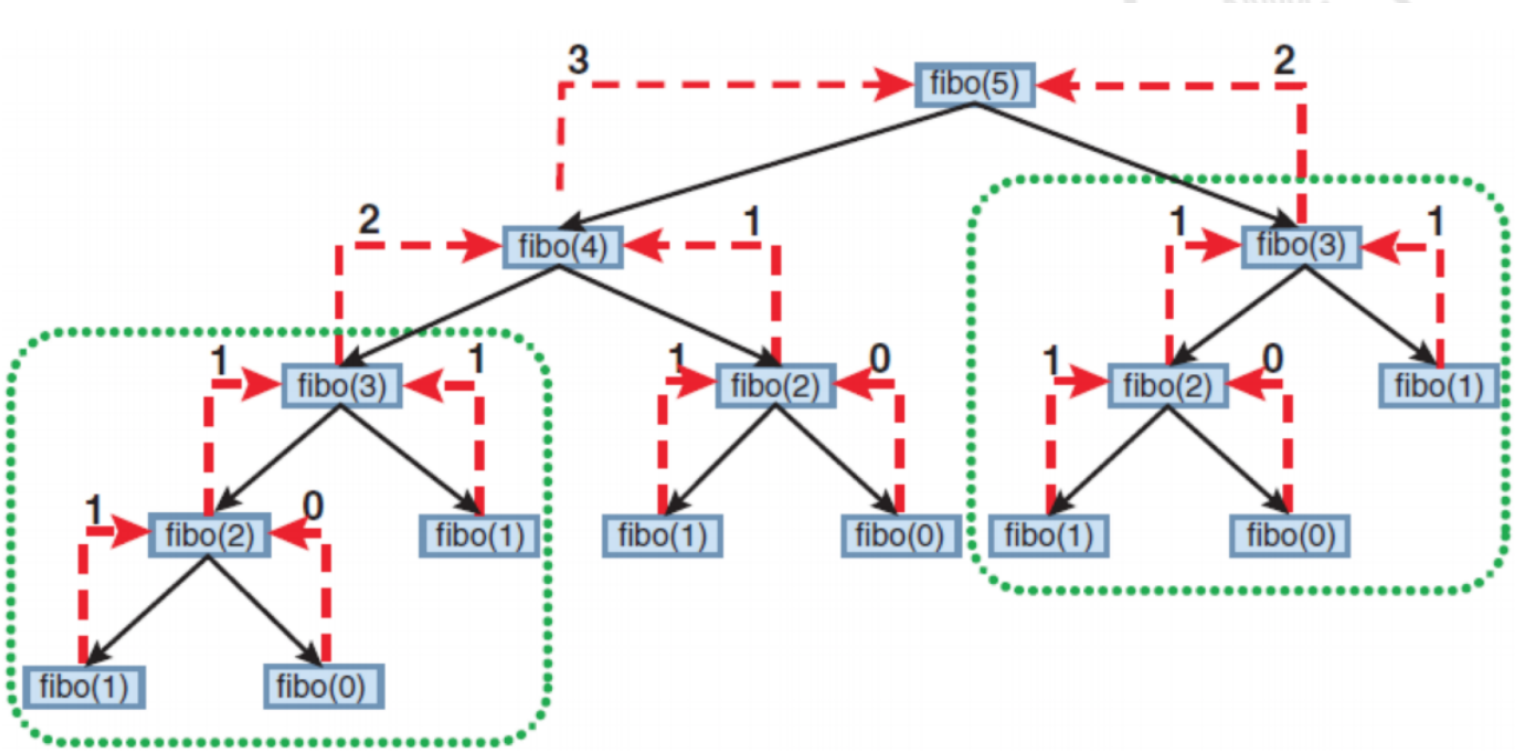
- Apesar de elegante, não é eficiênte



DEPARTAMENTO DE COMPUTAÇÃO

Problema de Fibonacci

- Aumentado para 5



DEPARTAMENTO DE COMPUTAÇÃO

Referências

- André Luiz Villar Forbellone, Henri Frederico Eberspächer, **Lógica de programação** (terceira edição), Pearson, 2005, ISBN 9788576050247.
- Ulysses de Oliveira, **Programando em C** - Volume I - Fundamentos, editora Ciência Moderna, 2008, ISBN 9788573936599
- **Slides baseados no material do site "Linguagem C Descomplicado"**
  - `https://programacaodescomplicada.wordpress.com/complementar/`

DEPARTAMENTO DE COMPUTAÇÃO