

Algoritmos e Programação Estruturada

Funções e Recursividade

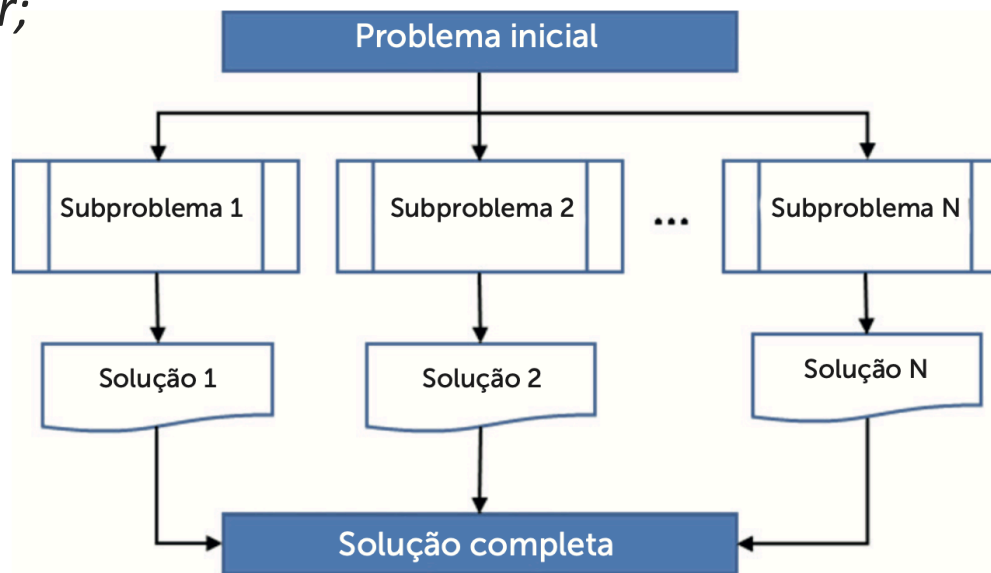
Ma. Vanessa Matias Leite

- Unidade de Ensino: 03
- Competência da Unidade: Conhecer e compreender o que são e como utilizar funções e procedimentos dentro de uma linguagem de programação
- Resumo: Saber utilizar funções e procedimentos para implementar soluções computacionais que utilizem tais recursos
- Palavras-chave: Função; parâmetro; recursividade; escopo; ponteiro;
- Título da Teleaula: Funções e Recursividade
- Teleaula nº: 03

Procedimentos e Funções

Funções

A ideia de criar programas com blocos de funcionalidades vem de uma técnica de projeto de algoritmos chamada *dividir para conquistar*;



Fonte: Scheffer (2018)

Função

Trecho de código escrito para solucionar um subproblema;

- Dividir a complexidade de um problema maior;
- Evitar repetição de código;
- Modularização;

Funções

```
<tipo de retorno> <nome> (<parâmetros>)  
{  
    <Comandos da função>  
    <Retorno> (não obrigatório)  
}
```

Funções

<tipo de retorno> – Obrigatório. Esse parâmetro indica qual o tipo de valor a função irá retornar. Pode ser um valor inteiro (*int*), decimal (*float* ou *double*), caractere (*char*), etc.

<nome> – Obrigatório. Parâmetro que especifica o nome que identificará a função.

Funções

<parâmetros> – Opcional.

<retorno> – Quando o tipo de retorno for *void* esse parâmetro não precisa ser usado, porém, quando não for *void* é obrigatório.

Funções

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

2.  int  somar(){
3.      return 2 + 3;
4.  }

5.  int main(){
6.      int resultado = 0;
7.      resultado = somar();
8.      printf("O resultado da funcao e = %d",re-
9. sultado);
10.     return 0;
11. }
```

Fonte: Scheffer (2018)

Funções com Ponteiros

Ponteiro

Tipo especial de variável, que armazena um endereço de memória;

O acesso à memória é feito usando dois operadores:

- Asterisco (*): usado para criação do ponteiro;
- “&” : Acessar o endereço da memória;

Ponteiro

`<tipo> *<nome_do_ponteiro>;`

Exemplo: **`int *idade;`**

A criação de um ponteiro só faz sentido se for associada a algum endereço de memória.

1. `int ano = 2018;`

2. `int *ponteiro_para_ano = &ano;`

Função com ponteiros

Função que retorna um vetor:

`int[10] calcular()`  ERRADO!

O **correto** para este caso é utilizar ponteiros.

```
tipo* nome() {  
    tipo vetor[tamanho];  
    return vetor;  
}
```

Fonte: Scheffer (2018)

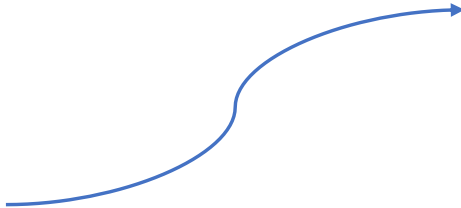
Funções com ponteiros

```
#include<stdio.h>

int* gerarRandomico() {

    static int r[10];
    int a;

    for(a = 0; a < 10; ++a) {
        r[a] = rand();
        printf("r[%d] = %d\n", a, r[a]);
    }
    return r;
}
```



```
r[0] = 41
r[1] = 18467
r[2] = 6334
r[3] = 26500
r[4] = 19169
r[5] = 15724
r[6] = 11478
r[7] = 29358
r[8] = 26962
r[9] = 24464
```

Fonte: Scheffer (2018)

Função *malloc()*

- Alocar memória dinamicamente;
- Exemplo: `int* memoria = malloc (100);`
- Retorna dois valores: NULL ou um ponteiro genérico;

Escopo e Passagem de Parâmetros

Escopo

```
int testar() {  
    int x = 10;  
    return x;  
}  
  
int main() {  
    int x = 20;  
    printf("\n Valor de x na funcao main()  
%d",x);  
    printf("\n Valor de x na funcao testar()  
%d",testar());  
  
    return 0;  
}
```

Fonte: Scheffer (2018)

Escopo

Variável Local: são “enxergadas” somente dentro do corpo da função onde foram definidas;

Variável Global: criá-la fora da função, assim ela será visível por todas as funções do programa. Geralmente adota-se declará-las após as bibliotecas.

Escopo

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

```
int x = 10;
```

```
int testar(){  
    return 2*x;  
}
```

```
int main(){  
    printf("\n Valor de x global = %d",x);  
    printf("\n Valor de x global alterado na funcao  
testar() = %d",testar());  
  
    return 0;  
}
```

Fonte: Scheffer (2018)

Parâmetros

<tipo de retorno> <nome> (<parâmetros>)

{

 <Comandos da função>

 <Retorno> (não obrigatório)

}

Parâmetros

Passagem de Valor: a função cria variáveis locais automaticamente para armazenar esses valores e após a execução da função essas variáveis são liberadas.

```
#include<stdio.h>

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

int main(){
    int result;
    result = somar(10,15);
    printf("\n Resultado da soma = %d",result);

    return 0;
}
```

Passagem por referência

- Ponteiro e endereço de memória;
- Não será criada uma cópia dos argumentos passados;
- Será passado o endereço da variável e função trabalhará diretamente com os valores ali armazenados;

Cálculo de Massa

Foi solicitado a você automatizar o cálculo de uma reação chamada de proteção. Nessa reação, um composto A, de massa 321,43 g/mol será somando a um composto B de massa 150,72 g/mol. Seu programa, além de calcular o composto com base nas informações do usuário, deverá também exibir os valores de referência das combinações: (1,2 : 1,0), (1,4 : 1,0) e (1,0 : 1,6).

Exercício

() Asterisco (*) é usado para criação do ponteiro e o “&” é usado para acessar o endereço da memória;

() A função busca dividir a complexidade de um problema maior e evitar repetição de código;

() A passagem de parâmetro não uma cópia dos argumentos passados;

Recursividade

Função Recursiva

A função será invocada dentro dela mesma.

```
<tipo> funcaoRecursiva(){  
    //comandos  
    funcaoRecursiva();  
    //comandos  
}  
  
void main(){  
    //comandos  
    funcaoRecursiva();  
    //comandos  
}
```

Chamando a si próprio

Fonte: Scheffer (2018)

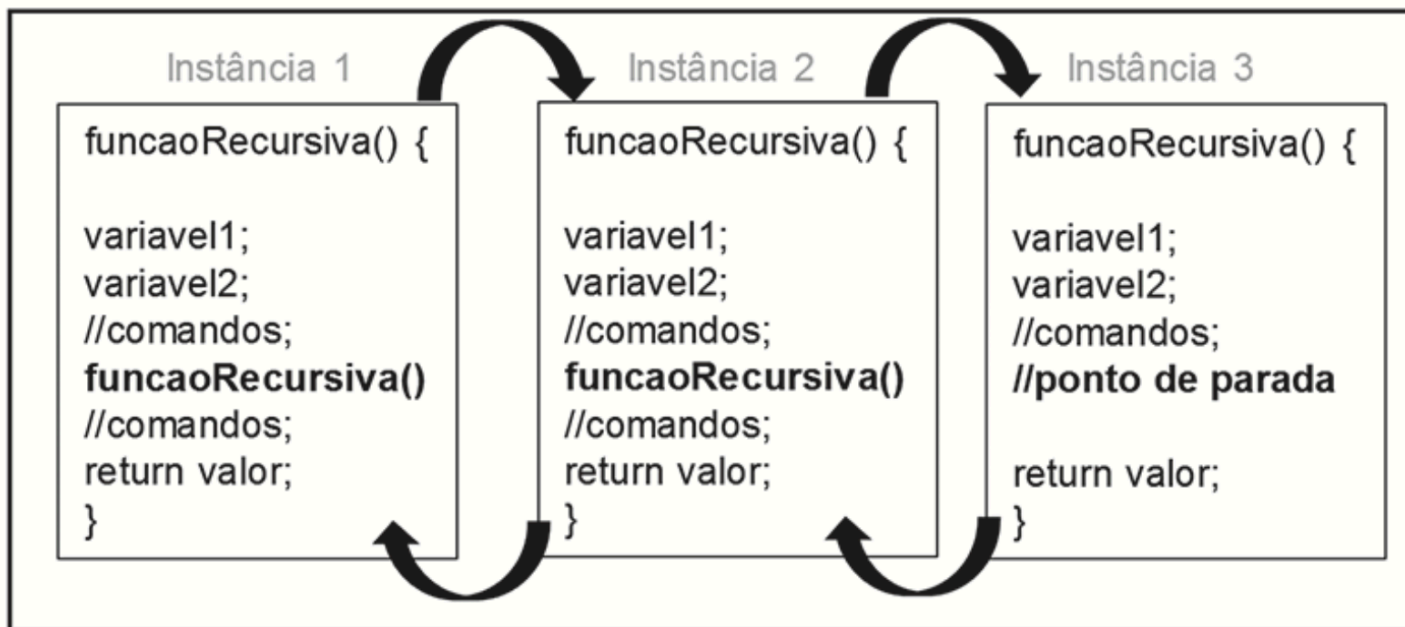
Função Recursiva

Um algoritmo recursivo resolve um problema dividindo-o em subproblemas mais simples, cujo a solução é a aplicação dele mesmo.

- Ponto de Parada;
- Variáveis na memória de trabalha;
- As variáveis são independentes;

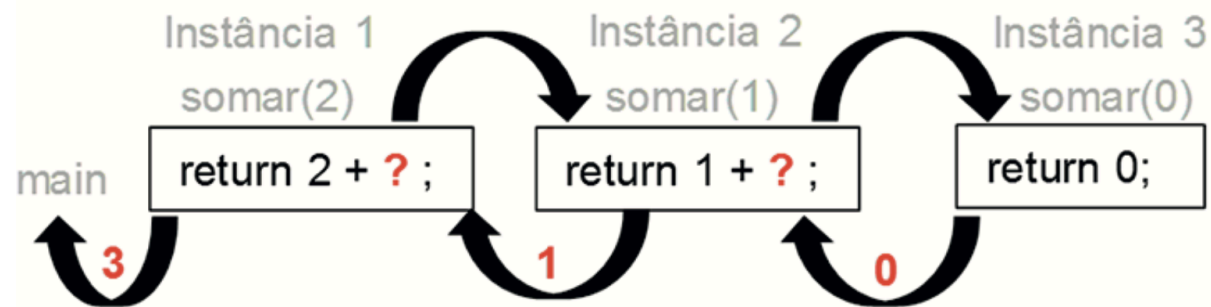
Função Recursiva

Memória de trabalho



Fonte: Scheffer (2018)

```
int somar(int valor){  
    if(valor != 0){  
        return valor + somar(valor-1);  
    }  
    else{  
        return valor;  
    }  
}
```



Fonte: Scheffer (2018)

Exemplos de Recursividade

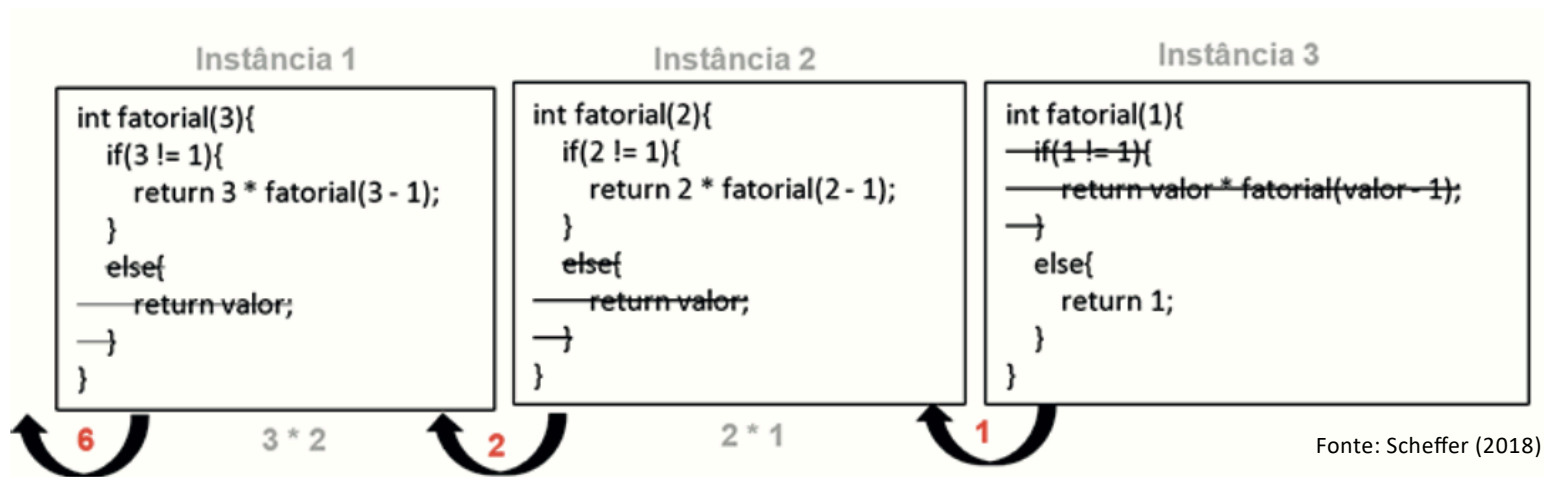
Fatorial

O fatorial de um número qualquer N consiste em multiplicações sucessivas até que N seja igual ao valor unitário, ou seja, $5! = 5 \times 4 \times 3 \times 2 \times 1$, que resulta em 120.

Fatorial

```
int fatorial(int valor){  
    if(valor != 1){  
        return valor * fatorial(valor - 1);  
    }  
    else{  
        return valor;  
    }  
}
```

Fatorial



Fibonacci

A sequência de Fibonacci:

$$F_n = \begin{cases} 0 & \text{if } n = 0 \\ 1 & \text{if } n = 1 \\ F_{n-1} + F_{n-2} & \text{if } n > 1 \end{cases}$$

Encontrar o n-ésimo elemento da sequência.

Fibonacci

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

Fibonacci

```
int Fib(int n)
{
    int i, j, f;
    i = 1; f = 0;
    for (j = 1; j <= n; j++) {
        f += i;
        i = f - i; }
    return f;
}
```

Recursividade

Foi solicitado a você implementar o método de Newton para o cálculo da raiz quadrada, porém, usando funções recursivas.

$$x_n = \frac{x_{n-1}^2 + n}{2x_{n-1}}$$

Você precisa solicitar ao usuário um número;

Você também deve especificar um valor inicial para a raiz e um critério de parada;

```
#include<stdio.h>
#include<math.h>
float calcularRaiz(float n, float raizAnt)
{
    float raiz = (pow(raizAnt, 2) + n)/(2 * raizAnt);
    if (fabs(raiz - raizAnt) < 0.001)
        return raiz;
    return CalcularRaiz(n, raiz);
}
```

```
void main() {  
    float numero, raiz;  
    printf("\n Digite um número para calcular a raiz: ");  
    scanf("%f", &numero);  
    raiz = calcularRaiz(numero, numero/2);  
    printf("\n Raiz quadrada funcao = %f", raiz);  
}
```

Exemplo em C

**Quando usar a
recursividade ?**

Recapitulando

Recapitulando

- Funções;
- Funções com parâmetros;
- Escopo e Passagem de Parâmetros;
- Recursividade;

