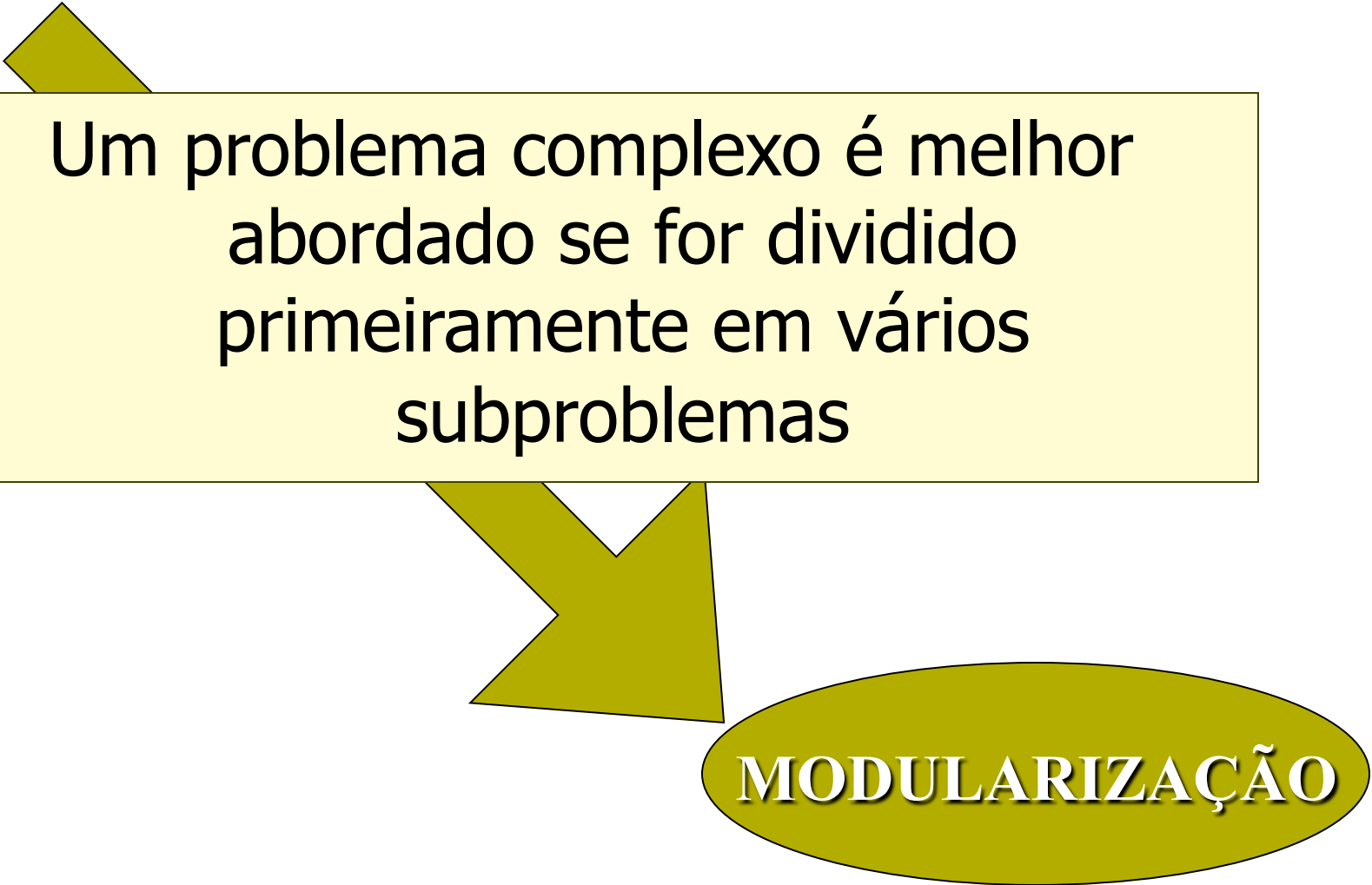




Modularização de Programas

SSC 304 - Introdução à Programação para Engenharias
Profa. Dra. Elisa Yumi Nakagawa

MODULARIZAÇÃO



Um problema complexo é melhor
abordado se for dividido
primeiramente em vários
subproblemas

MODULARIZAÇÃO

MODULARIZAÇÃO

The diagram illustrates the concept of modularization. At the top, the word 'MODULARIZAÇÃO' is written in a large, brown, stylized font. A green arrow points from this title down to a yellow rectangular box containing the text 'Um problema complexo é melhor abordado se for dividido primeiramente em vários subproblemas'. From the bottom of this box, a large green arrow points down to a brown rectangular box labeled 'MÓDULOS'. To the right of the 'MÓDULOS' box, a green oval contains the word 'MODULARIZAÇÃO' in white, bold, serif font. A yellow arrow points from the 'MÓDULOS' box towards this oval, indicating that modules are the result of the modularization process.

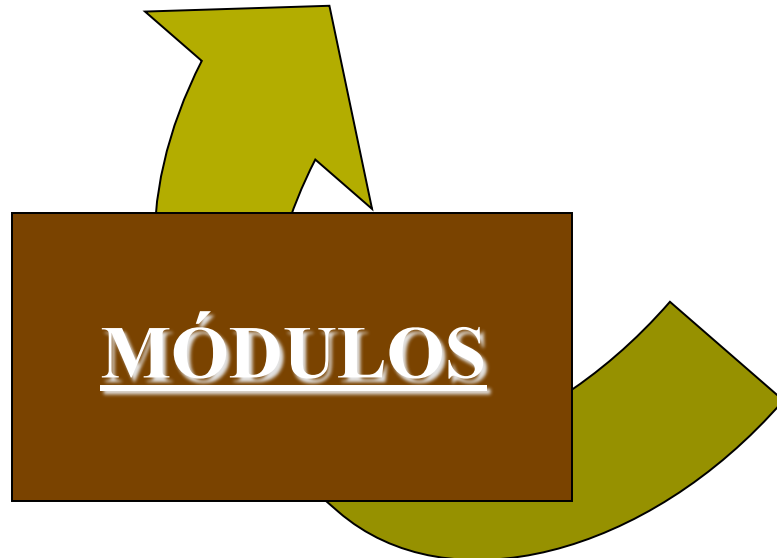
Um problema complexo é melhor
abordado se for dividido
primeiramente em vários
subproblemas

MÓDULOS

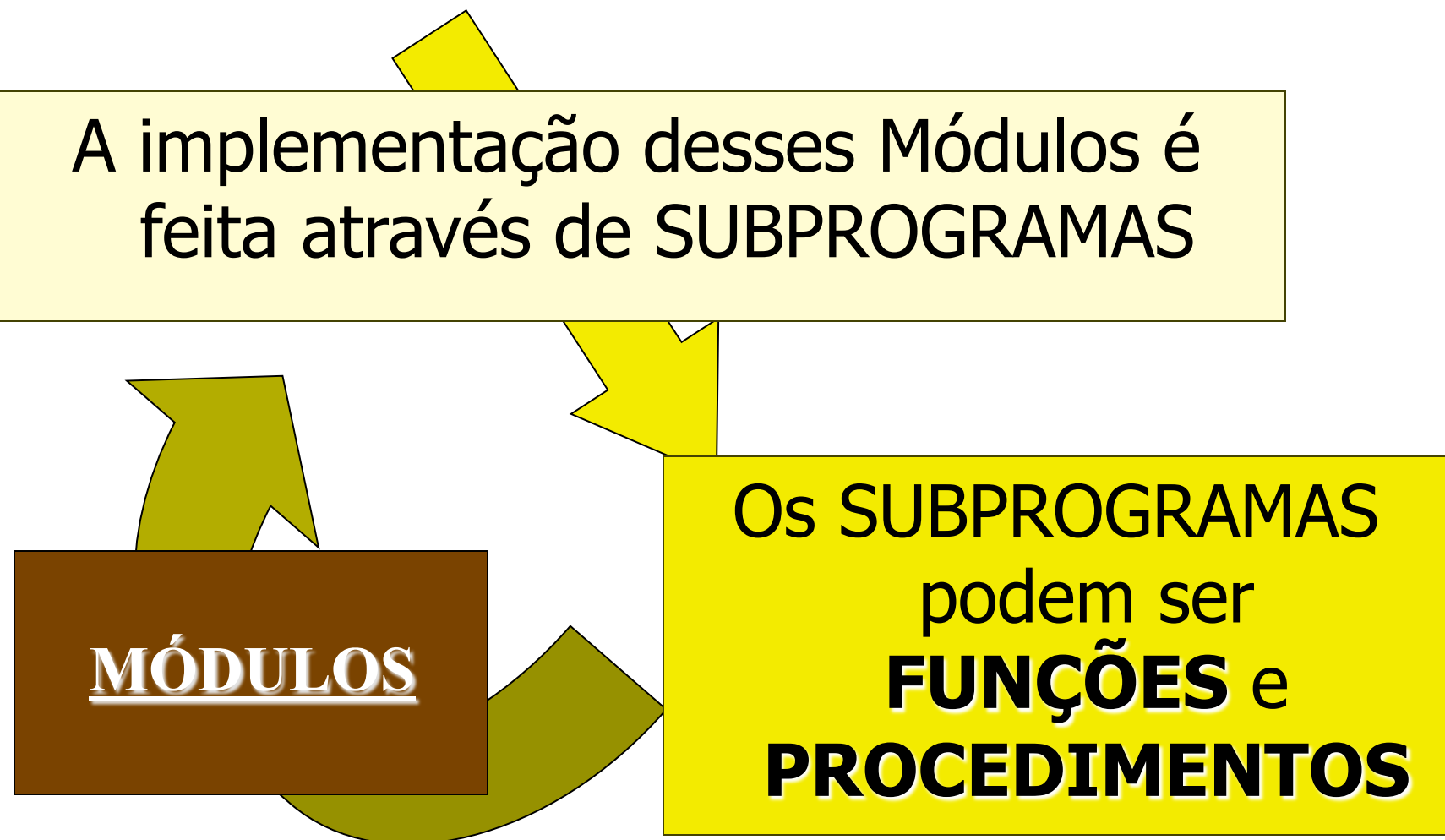
MODULARIZAÇÃO

MODULARIZAÇÃO

A implementação desses Módulos é feita através de SUBPROGRAMAS



MODULARIZAÇÃO



A implementação desses Módulos é feita através de SUBPROGRAMAS

MÓDULOS

Os SUBPROGRAMAS
podem ser
FUNÇÕES e
PROCEDIMENTOS

SUBPROGRAMAS

- SUBPROGRAMAS devem realizar uma única funcionalidade no programa.
 - possuem “código independente” e devem ser definidos separadamente no programa
 - Na linguagem C, a definição do subprograma (função) deve ocorrer antes da função que irá utilizá-lo.
- SUBPROGRAMAS são declarados no início do programa
 - Prototipação de funções

Estrutura de uma função

```
tipodafunção nomedafunção (tipo  
    var1, ..., tipo varn)
```

inicio

declaração de variáveis locais

corpo da função

```
retorne (variável de retorno) ;
```

Fim.

Prototipação da função

Declaração da função:

```
tipodafunção nomedafunção (tipol, ...,  
    tipon) ;
```

Estrutura de uma função

- Retorna um **único** valor
- Na definição da função devem ser declarados:
 - o **tipo** de todos os **parâmetros** (argumentos)
 - o **tipo** do valor que a função **retorna**
 - todas as **variáveis** utilizadas internamente no subprograma (variáveis locais)
- Para utilizar a função no programa principal, basta colocar seu **nome** (identificador) e os valores para os parâmetros.

DEFINIÇÃO DE FUNÇÃO

Exemplo

- Dados dois números N e K , calcular a Combinação

$$C_{N,K} = \frac{N!}{K!(N-K)!}$$

- Com a definição de uma função **fat (X)** que calcula o fatorial de um dado X , o cálculo da combinação fica:

$$CNK \leftarrow \mathbf{fat (N)} / (\mathbf{fat (K)} * \mathbf{fat (N-K)})$$

Algoritmo fat

inteiro fat(inteiro x)

inicio

inteiro i, f=1;

para i=x até 1 faça

f=f*i;

fim-para;

retorna (f);

fim.

Algoritmo principal.

inicio.

inteiro n, k, c;

leia(n, k);

c = **fat**(n) / (**fat**(k) * **fat**(n-k)) ;

escreva(c);

fim.

Escopo de variáveis

- Significa a visibilidade de uma variável perante os diversos subprogramas integrantes do programa (ou algoritmo)
 - **Variável global:** declarada no início do algoritmo ou programa (fora dos subprogramas e programa principal) e é visível por todos.
 - **Variável local:** declarada dentro de um subprograma e somente visível dentro do mesmo.

Algoritmo troca

troca(inteiro a, inteiro b)

inicio

inteiro aux;

aux = a;

a = b;

b = aux;

fim.

Algoritmo principal.

inicio.

inteiro a, b;

a = 10; b = 20;

troca(a,b) ;

escreva(a,b) ;

fim.

Algoritmo troca

troca(inteiro a, inteiro b)

inicio

inteiro aux;

aux = a;

a = b;

b = aux;

fim.

a, b e aux são
variáveis locais
de troca()

Algoritmo principal.

inicio.

inteiro a, b;

a = 10; b = 20;

troca(a,b) ;

escreva(a,b) ;

fim.

a e b são
variáveis locais
do alg. princ.

Algoritmo troca

troca(inteiro a, inteiro b)

inicio

inteiro aux;

aux = a;

a = b;

b = aux;

fim.

Algoritmo principal.

inicio.

inteiro a, b;

a = 10; b = 20;

troca(a,b) ;

escreva(a,b) ;

fim.

a, b e aux são
variáveis locais
de troca()

Não realiza a troca!

Não é possível retornar
dois valores

Argumentos são
passados por valor

a e b são
variáveis locais
do alg. princ.

Algoritmo troca

inteiro a, b;

troca()

inicio

inteiro aux;

aux = a;

a = b;

b = aux;

fim.

a e b são
variáveis
globais

aux é a única
variável local

Realiza a troca com o
uso de variável global

Algoritmo principal.

inicio.

a = 10; b = 20;

troca();

escreva(a,b);

fim.

Não é uma boa
solução. Mais a frente
iremos ver uma
solução melhor!!!

Escopo de variáveis

- A variável global existe durante toda a execução do programa
 - Interessante quando a mesma é utilizada por várias funções
 - Muitas variáveis globais podem prejudicar a execução do programa (overflow)
 - Os parâmetros enviados para as funções ajudam no entendimento da finalidade da função
 - O uso de variável global deve ser analisado.



Como Transformar um Programa em um Subprograma

Programa para verificar se um número é primo

Algoritmo vprimo
início.

inteiro n, c, i;

leia (n);

c = 1; *//true*

i = 2; *//divisor do número*

enquanto ((i < n) E c)então

se (n % i == 0) então *//resto da divisão for zero*

c = 0;

senão

i = i + 1;

fim-se;

fim-enquanto;

se (c) então

escreva("O Numero e Primo");

senão

escreva("O Numero nao e Primo");

fim-se;

fim do algoritmo.

Procura por um
divisor diferente
de 1 e do próprio
número

Programa para verificar se um número é primo

Algoritmo vprimo

início.

inteiro n, c, i;

leia (n);

c = 1; *//true*

i = 2; *//divisor do número*

enquanto ((i < n) E c)então

se (n % i == 0) então *//resto da divisão for zero*

c = 0;

senão

i = i + 1;

fim-se;

fim-enquanto;

se (c) então

escreva("O Numero e Primo");

senão

escreva("O Numero nao e Primo");

fim-se;

fim do algoritmo.

Este trecho do
programa
verifica se o
número é primo

OBJETIVO
PRINCIPAL

Programa para verificar se um número é primo

Algoritmo vprimo
início.

inteiro n, c, i;
leia (n);

c = 1; *//true*

i = 2; *//divisor do número*

enquanto ((i < **n**) E c)então

se (**n** % i == 0) então *//resto da divisão for zero*

c = 0;

senão

i = i + 1;

fim-se;

fim-enquanto;

se (c) então

escreva("O Numero e Primo");

senão

escreva("O Numero nao e Primo");

fim-se;

fim do algoritmo.

Corpo da
função

n : parâmetro de
entrada

c: saída

Programa para verificar se um número é primo

Algoritmo vprimo
início.

inteiro n, c, i;
leia (n);

c = 1; *//true*

i = 2; *//divisor do número*

enquanto ((i < **n**) E c)então

se (**n** % i == 0) então *//resto da divisão for zero*

c = 0;

senão

i = i + 1;

fim-se;

fim-enquanto;

se (c) então

escreva("O Numero e Primo");

senão

escreva("O Numero nao e Primo");

fim-se;

fim do algoritmo.

A leitura de **n** fica no
programa principal

As mensagens de
saída ficam no
programa principal

Programa para verificar se um número é primo

```
inteiro primo(inteiro n)
inicio.
inteiro c, i;
c = 1; //true
i = 2; //divisor do número
enquanto ( (i < n ) E c )então
    se (n % i == 0) então //resto da divisão for zero
        c = 0;
    senão
        i = i + 1;
    fim-se;
fim-enquanto;
retorne(c);
fim.
```

```
algoritmo principal
inicio.
inteiro n;
leia (n);
se primo(n) então
    escreva("O Numero e Primo" );
senão
    escreva("O Numero nao e Primo");
fim-se;
fim.
```

Algoritmo vprimo
declarações variáveis globais

inteiro primo(**n**)

fim

algoritmo principal

inicio.

inteiro n;

leia (n);

se primo(n) então

escreva("O Numero e Primo");

senão

escreva("O Numero nao e

Primo");

fim-se;

fim.

inteiro primo(inteiro n)

inicio.

inteiro c, i;

c = 1; *//true*

i = 2; *//divisor do número*

enquanto ((i < **n**) E c)então

se (**n** % i == 0) então *//resto da divisão for zero*

c = 0;

senão

i = i +1;

fim-se;

fim-enquanto;

retorne(**c**);

fim.

ALGORITMO

Exercício

1. Faça um algoritmo que lê um valor inteiro e positivo n e um algarismo d ($0 \leq d \leq 9$) e escreve quantas vezes d aparece em n . Utilize uma função para fazer a contagem.
2. Fazer um algoritmo para calcular a raiz quadrada de um número positivo, usando o roteiro abaixo, baseado no método de aproximações sucessivas de Newton:
 - a. Seja y o número:
 - i. a primeira aproximação para a raiz quadrada de y é $x_1 = y/2$;
 - ii. as sucessivas aproximações serão $x_{n+1} = (x_n^2 + y) / 2x_n$
 - b. O algoritmo deverá prever 20 aproximações