

BCC701 – Programação de Computadores I
Universidade Federal de Ouro Preto
Departamento de Ciência da Computação

www.decom.ufop.br/bcc701
2012/01



Semana 05:

Tipos de Dados.

Uso de Contadores.

Comandos de Repetição/Iteração.

Material Didático Unificado.

Agenda

- Tipos de dados;
- Uso de contadores;
- Comandos de repetição/iteração;
- Exercícios.

Tipos de dados;

Uso de contadores;

Comandos de repetição/iteração;

Exercícios.

TIPOS DE DADOS

Introdução

- “**Tipos de dados**” em linguagem de programação definem a natureza do dado armazenado em uma variável ou manipulado nas operações;
- **Tipos de dados primitivos:** tipos de dados básicos fornecidos pela linguagem, no Scilab:
 - Número;
 - Booleano (lógico);
 - *String* (texto);
- Normalmente uma linguagem oferece também tipos mais complexos, no Scilab: vetores, matrizes, registros, entre outros;
 - Alguns destes tipos são abordados ao longo da disciplina, neste momento o foco está nos tipos primitivos.

Introdução

- O Scilab é uma “**linguagem dinamicamente tipada**”;
- Isto significa que:
 - Os tipos são definidos em tempo de execução;
 - Não é necessário definir explicitamente os tipos das variáveis;
 - As variáveis podem mudar de tipo ao longo do programa;
 - Os tipos são definidos pelo uso, por exemplo, nas atribuições:
 - --> A = 2; // Neste caso A será um valor numérico
 - --> A = 2.5; // A continua numérico
 - --> A = “texto”; // Agora A é do tipo *string* (texto)
 - O tipo também é definido pelo resultado de uma expressão:
 - --> A = 2 >= 2.5 // A não será um número, mas um booleano (%f)
A =
F
-->

Números

- O Scilab não diferencia tipos numéricos, como: Inteiro, Real ou Complexo;
- O valor armazenado e o uso de funções específicas é que caracterizará o “tipo numérico”;
- A seguir algumas funções para manipulação de números.

Números

Número

Booleano (lógico)

String (texto)

- Algumas funções para manipulação de números **inteiros**:

Função	Descrição	Exemplos
int(X)	Arredondamento de X na direção de 0 (zero). <i>Em outras palavras:</i> parte inteira do valor de X .	$\text{int}(2) = 2$ $\text{int}(2.3) = 2$ $\text{int}(2.8) = 2$ $\text{int}(-2.8) = -2$
ceil(X)	Arredondamento de X na direção de <i>mais infinito</i> . <i>Em outras palavras:</i> menor valor inteiro maior ou igual ao número X .	$\text{ceil}(2) = 2$ $\text{ceil}(2.3) = 3$ $\text{ceil}(2.8) = 3$ $\text{ceil}(-2.8) = -2$
floor(X)	Arredondamento de X na direção de <i>menos infinito</i> . <i>Em outras palavras:</i> maior valor inteiro menor ou igual ao número X .	$\text{floor}(2) = 2$ $\text{floor}(2.3) = 2$ $\text{floor}(2.8) = 2$ $\text{floor}(-2.8) = -3$
round(X)	Arredondamento para o inteiro mais próximo.	$\text{round}(2) = 2$ $\text{round}(2.3) = 2$ $\text{round}(2.8) = 3$ $\text{round}(-2.8) = -3$

Números

Tipos Primitivos do Scilab

Número

Booleano (lógico)

String (texto)

- Algumas funções para manipulação de números **complexos**:
 - Lembre-se:
 - A unidade imaginária é representada por %i (constante igual a $\sqrt{-1}$);
 - A declaração de um número complexo é feita com o uso desta constante, como por exemplo: $A = 3 + 4 * \%i$, ou $B = 5 - 6 * \%i$;
 - As operações matemáticas também funcionam, exemplo: $C = A - B$;
 - C será igual a $-2 + 10 * \%i$.

Função	Descrição	Exemplos
real(X)	Parte real de X.	$\text{real}(A) = 3$ $\text{real}(B) = 5$ $\text{real}(C) = -2$
imag(X)	Parte imaginária de X.	$\text{imag}(A) = 4$ $\text{imag}(B) = -6$ $\text{imag}(C) = 10$
conj(X)	Conjugado de X.	$\text{conj}(A) = 3 - 4 * \%i$ $\text{conj}(B) = 5 + 6 * \%i$ $\text{conj}(C) = -2 - 10 * \%i$

Números

Número

Booleano (lógico)

String (texto)

- **Exemplo:** Escreva um programa que, dado um número de conta corrente com três dígitos, retorne o seu dígito verificador, que é calculado da seguinte maneira:

Número da conta: 235

- 1) Somar o número da conta com seu inverso: $235 + 532 = 767$
- 2) multiplicar cada dígito pela sua ordem posicional e somar estes resultados:

$$\begin{array}{r}
 7 \\
 \times 1 \\
 \hline
 7
 \end{array}
 +
 \begin{array}{r}
 6 \\
 \times 2 \\
 \hline
 12
 \end{array}
 +
 \begin{array}{r}
 7 \\
 \times 3 \\
 \hline
 21
 \end{array}
 = 40$$

- 3) o dígito verificador da conta é o último dígito ($40 \rightarrow 0$)

Números

Número

Booleano (lógico)

String (texto)

- **Exemplo:** Solução:

```
nroConta = input("DIGITE O NÚMERO DA CONTA: ");  
d1 = int( nroConta / 100 );  
d2 = int( modulo(nroConta, 100) / 10 );  
d3 = int( modulo (nroConta, 10) );  
inverso = int (d3 * 100 + d2 * 10 + d1);  
soma = nroConta + inverso;  
d1 = int( soma / 100 ) * 1;  
d2 = int( modulo(soma, 100) / 10 ) * 2;  
d3 = int( modulo (soma, 10) ) * 3;  
digitoV = int ( modulo( (d1 + d2 + d3), 10) );  
printf("\nO DÍGITO VERIFICADOR DA CONTA %g É %g", ...  
      nroConta, digitoV);
```

Três pontos (...) indica que o comando continua na próxima linha.

Booleano

- Como já vimos em aulas anteriores, valores booleanos podem assumir apenas dois valores:
 - **Verdadeiro:** %T ou %t;
 - **Falso:** %F ou %f;
- Expressões que envolvam operadores relacionais e lógicos sempre resultaram em um valor booleano, e são chamadas de expressões lógicas;
- Os comandos de decisão e iteração geralmente envolvem um valor booleano para determinar o fluxo de execução do programa.

String

- O Scilab também é capaz de manipular valores que não são numéricos e nem lógicos;
- **Valores textuais**, ou seja, que contém sequências de caracteres (letras, dígitos e outros símbolos, como #, \$, &, %, ?, !, @, <, ~, etc.), são chamados de **STRING**;
- Uma *string* deve ser **delimitada** por **aspas**;
 - No Scilab as aspas duplas (“) e as aspas simples (‘) são equivalentes, exemplos:
 - “Programação de Computadores”;
 - ‘Programação de Computadores’;
 - “Programação de Computadores”;
 - ‘Programação de Computadores”.

String

- Como inserir aspas em uma *string*?

```
--> x = 'String "com aspas"'
      !--error 276
```

Operador, comma, ou semicolon faltante.

```
--> x = 'String ""com aspas""'
```

```
x = String "com aspas"
```

```
--> x = "String 'com aspas'"
```

```
x = String 'com aspas'
```

```
--> x = 'String "'com aspas'"'
```

```
x = String "com aspas"
```

```
--> x = 'String "'com aspas'"'
```

```
x = String "com aspas"
```

String

- *Strings* podem ser concatenadas (justapostas):

```
--> a = "Programação";
```

```
--> b = " de ";
```

```
--> c = "Computadores";
```

```
--> d = a + b + c
```

```
d =
```

```
Programação de Computadores
```

```
-->
```

String

- **Atenção:** *Strings* formadas por dígitos não são considerados como valores numéricos, exemplo:

```
--> format(16)
```

```
--> %pi
```

```
%pi = 3.1415926535898
```

```
--> StringPI = "3.1415926535898"
```

```
StringPI = 3.1415926535898
```

```
--> 2 * %pi
```

```
ans = 6.2831853071796
```

```
--> 2 * StringPI
```

```
!--error 144
```

Operação indefinida para os dados operandos.

Verifique ou defina a função `%s_m_c` para overloading.

String

- **Atenção:** *Strings* formadas por dígitos não são considerados como valores numéricos, exemplo:

```
--> format(16)
--> %pi
%pi = 3.1415926535898
--> StringPI = "3.1415926535898"
StringPI = 3.1415926535898
--> 2 * %pi
ans = 6.2831853071796
--> 2 * StringPI
!--error 144
```

Números passam a ser exibidos com 16 posições (considerando o sinal e o ponto decimal).

format('e') define formato em notação científica (6.283185307E-01).

format('v') retorna ao formato padrão ("formato de variável").

Operação indefinida para os dados operandos.
Verifique ou defina a função `%s_m_c` para overloading.

String

- **Atenção:** *Strings* formadas por dígitos não são considerados como valores numéricos, exemplo:

```
--> format(16)
--> %pi
%pi = 3.1415926535898
--> StringPI = "3.1415926535898"
StringPI = 3.1415926535898
--> 2 * %pi
ans = 6.2831853071796
--> 2 * StringPI
!--error 144
```

Existe uma função que permite realizar esta operação:

```
--> 2 * eval(StringPI)
```

```
ans = 6.2831853071796
```

eval(StringPI) avalia a *string* como se fosse uma expressão, resultando um valor numérico ou lógico.

Operação indefinida para os dados operandos.
Verifique ou defina a função `%s_m_c` para overloading.

String

- Algumas funções para manipulação de *strings*:

Função	Descrição	Exemplos
convstr(<i>S</i>, <i>flag</i>)	Retorna os caracteres da <i>string S</i> convertidos para maiúscula (<i>flag</i> = 'u') ou minúscula (<i>flag</i> = 'l').	convstr('aBcD', 'u') convstr('aBcD', 'l')
length(<i>S</i>)	Comprimento em caracteres da <i>string S</i> .	length('abcd') length("Como usar part?")
part(<i>S</i>, <i>v</i>)	Extraí caracteres da <i>string S</i> em relação às posições definidas por <i>v</i> .	part("Como usar part?", 11:14) part("Como usar part?", [1:5, 11:14])
strindex(<i>S1</i>, <i>S2</i>)	Procura a posição da <i>string S2</i> na <i>string S1</i> .	strindex('aBcD', 'c') strindex('aBcD', 'd') strindex('aBcDc', 'c')
string(<i>N</i>)	Converte o número <i>N</i> em string.	string(10 + 5)
eval(<i>S</i>)	Retorna o valor numérico resultante da avaliação da string como uma expressão aritmética.	eval("10 + 20") eval("%pi") eval("cos(%pi)") eval("10 < 20")

Tipos de dados;

Uso de contadores;

Comandos de repetição/iteração;

Exercícios.

USO DE CONTADORES

Repetição

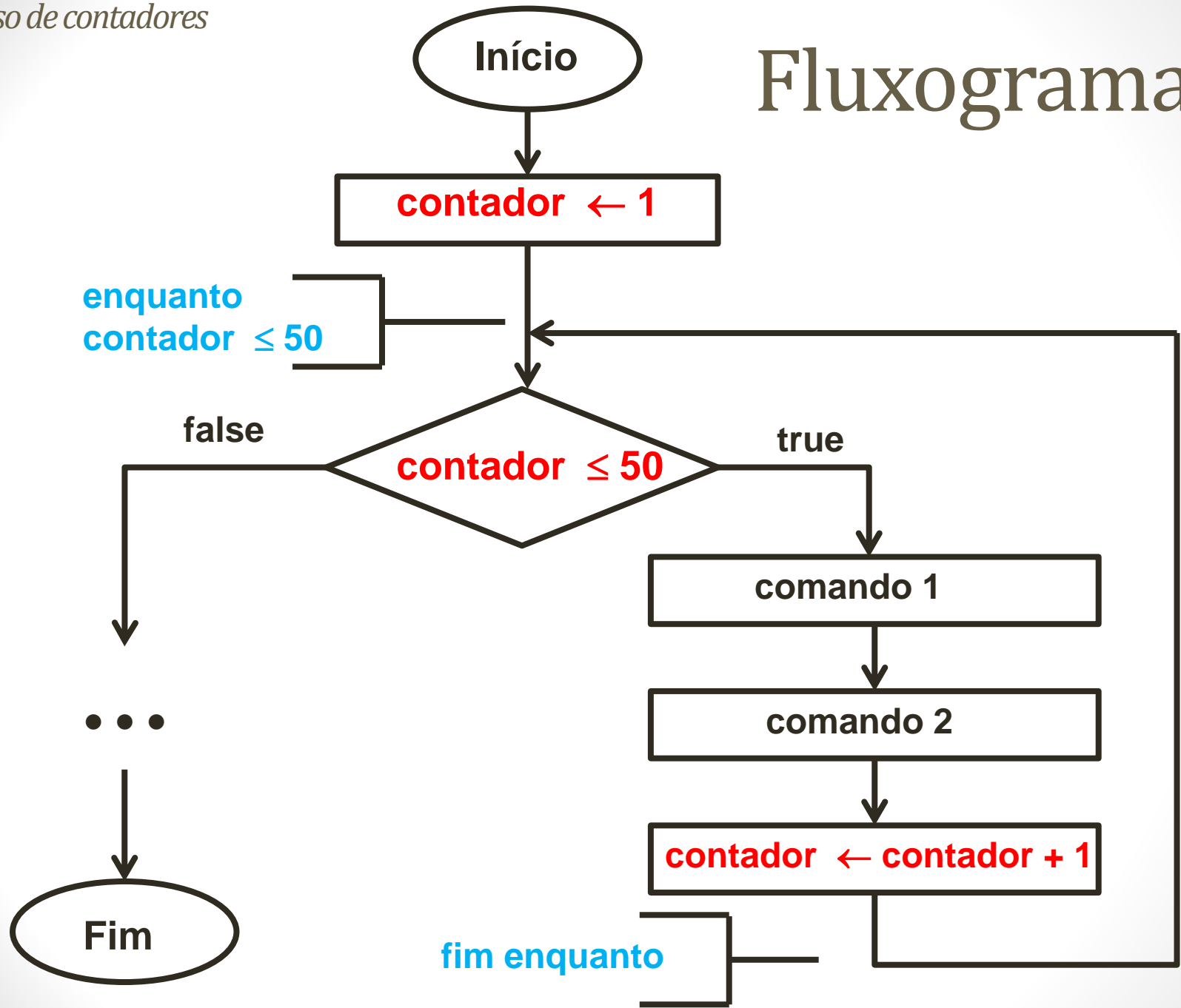
- Em determinadas aplicações é necessário executar repetidas vezes um bloco de comandos;
- A repetição do bloco de comandos deve ser finita, ou seja, o bloco deve ser repetido **n** vezes (valor limite);
- Para fazer este controle da repetição, utiliza-se uma **variável contadora (ou contador)**, que literalmente conta de 1 a **n** cada vez que o bloco é repetido;
- Um teste lógico assegura que as **n** repetições serão realizadas, comparando a cada execução o valor do contador com o limite das repetições.

Controle das Repetições

- O exemplo a seguir, ilustra o uso de um contador para controlar a repetição de um bloco de comandos 50 vezes;
- O bloco de comandos é composto por dois comandos quaisquer;
- Os comentários em um fluxograma são representados com uma descrição textual e o símbolo:



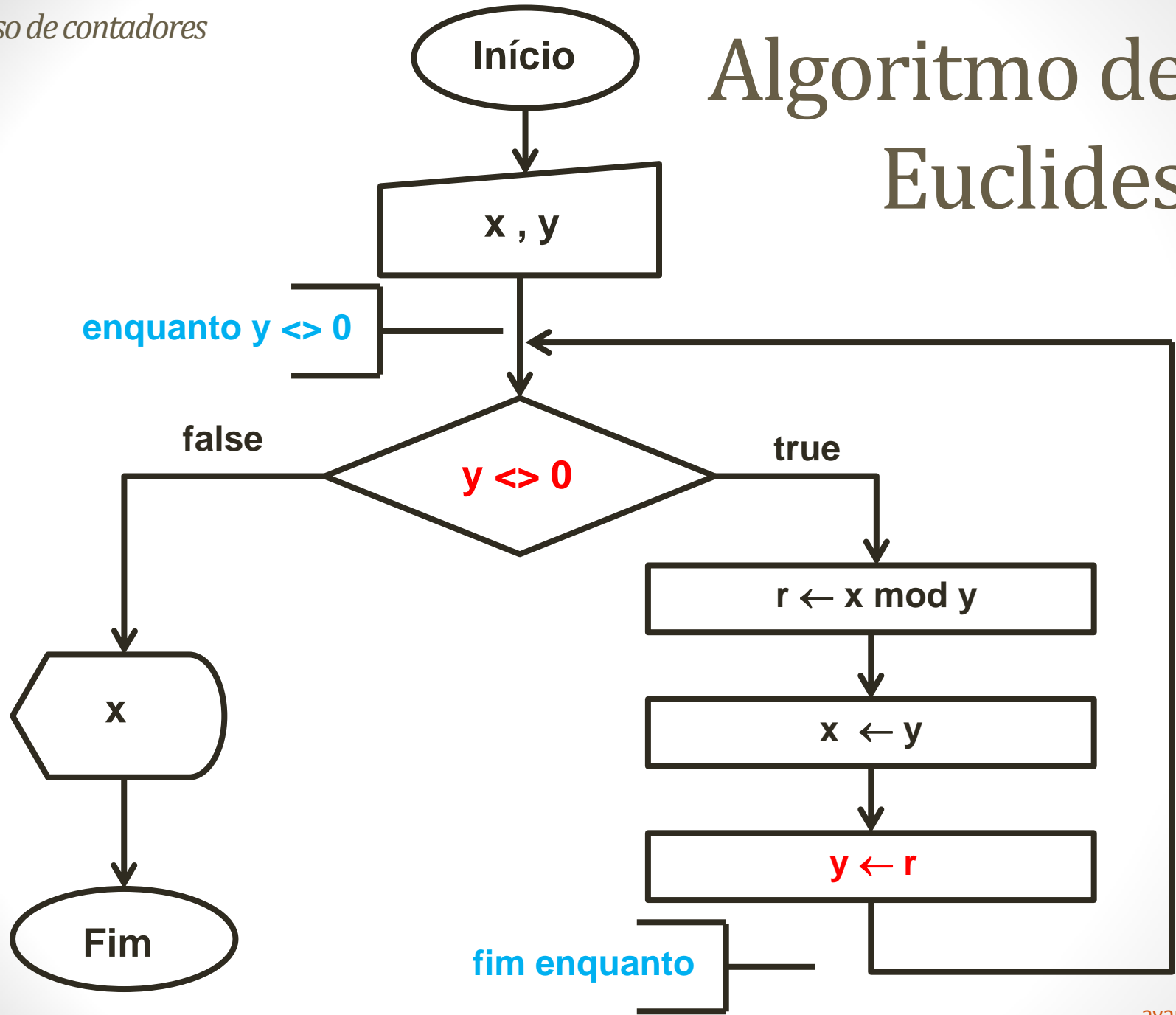
Fluxograma



Algoritmo de Euclides

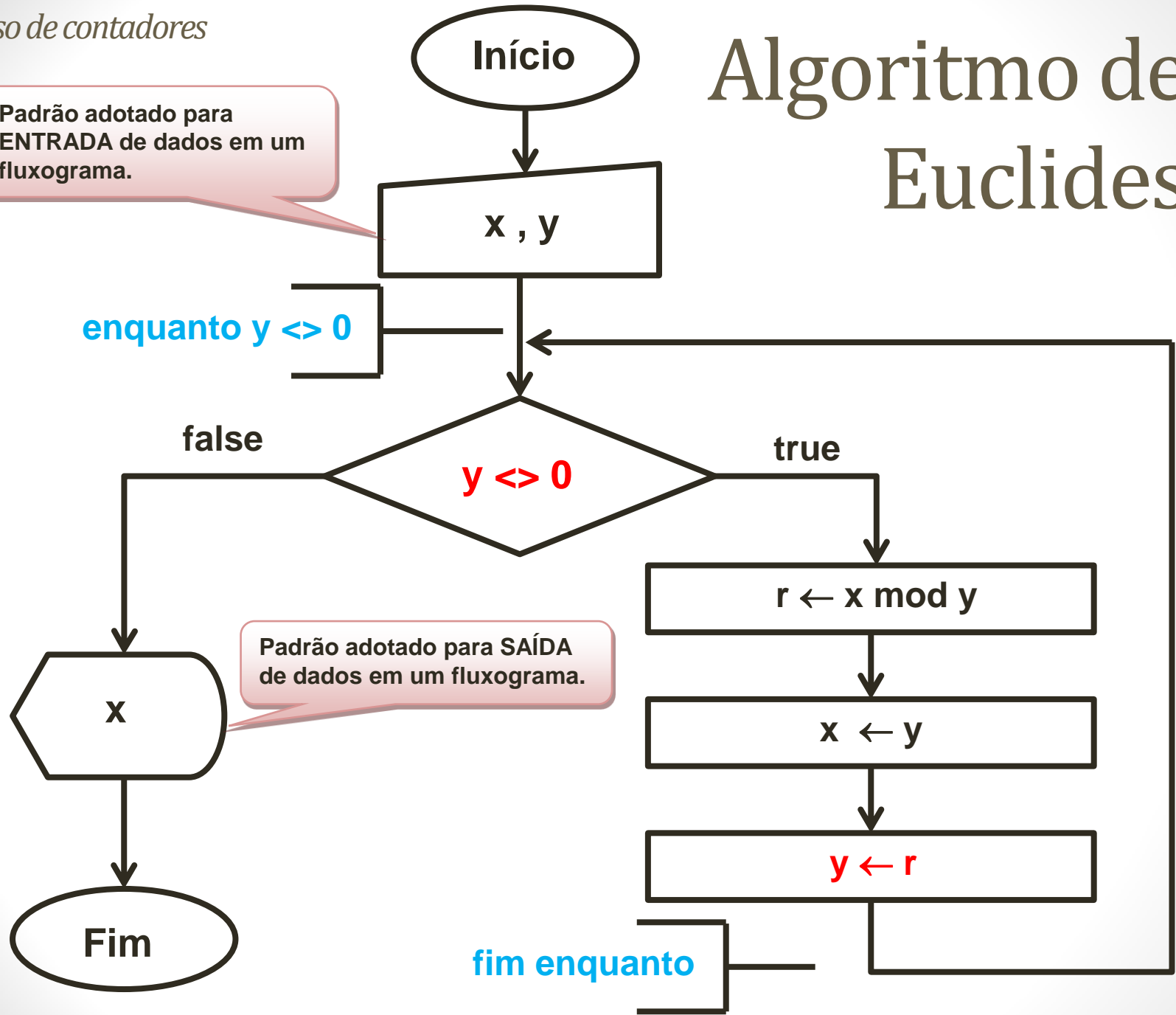
- O **algoritmo de Euclides** é utilizado para o cálculo do **MDC (Máximo Divisor Comum)** entre dois números inteiros;
- A seguir, o fluxograma:

Algoritmo de Euclides



Algoritmo de Euclides

Padrão adotado para ENTRADA de dados em um fluxograma.



Padrão adotado para SAÍDA de dados em um fluxograma.

Algoritmo de Euclides

- Supondo as entradas 544 e 119, vamos analisar o resultado:

x	y	r (resto)
544	119	68
119	68	51
68	51	17
51	17	0
17	0	0

- A resposta será **17** (o último valor atribuído a **x**);
- O quadro anterior é resultado de um “teste de mesa” (uma simulação da execução do programa feita à mão), a cada linha são definidos os valores das variáveis a cada iteração do laço.

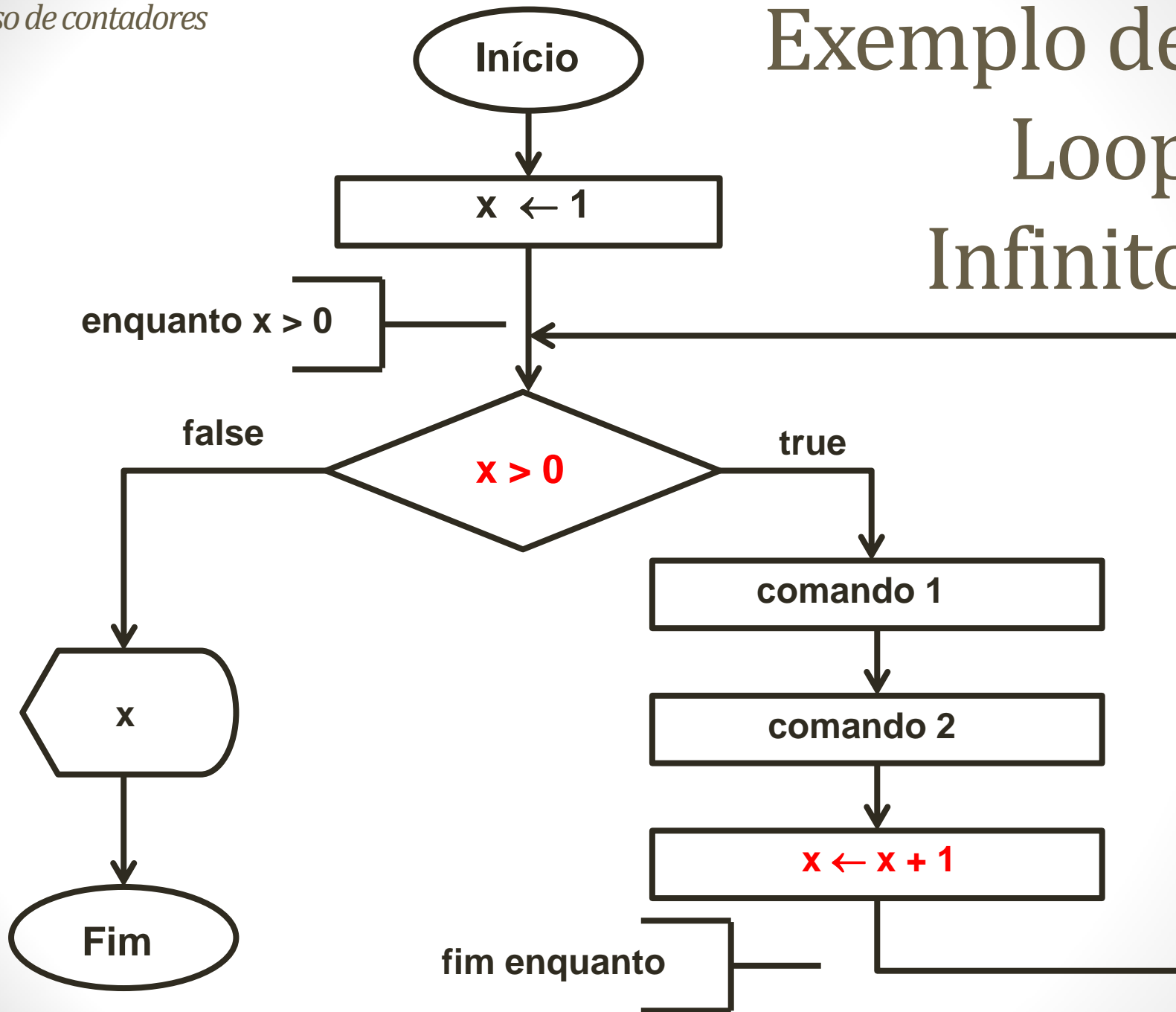
Observações no Fluxograma

- Aqui o símbolo do comando de decisão é utilizado com um significado diferente: ele é parte integral de um comando de repetição, servindo como um teste para indicar se os comandos em seu interior deverão ser executados novamente;
- A repetição é indicada pelo caminho fechado que sai do símbolo de decisão e que volta a ele;
- A expressão relacional escrita no símbolo de decisão, no caso de comando de repetição, representa um critério ou condição de parada da repetição.

Observações no Fluxograma

- Os comandos representados no caminho fechado representam os comandos que serão executados **enquanto** a **condição** (expressão relacional) for **verdadeira**;
- A condição de parada deve ser testada cuidadosamente, pois se estiver errada poderá levar a uma repetição infinita (**loop infinito**);
- No exemplo a seguir, nunca ocorrerá a impressão do valor da variável **x** na tela.

Exemplo de Loop Infinito

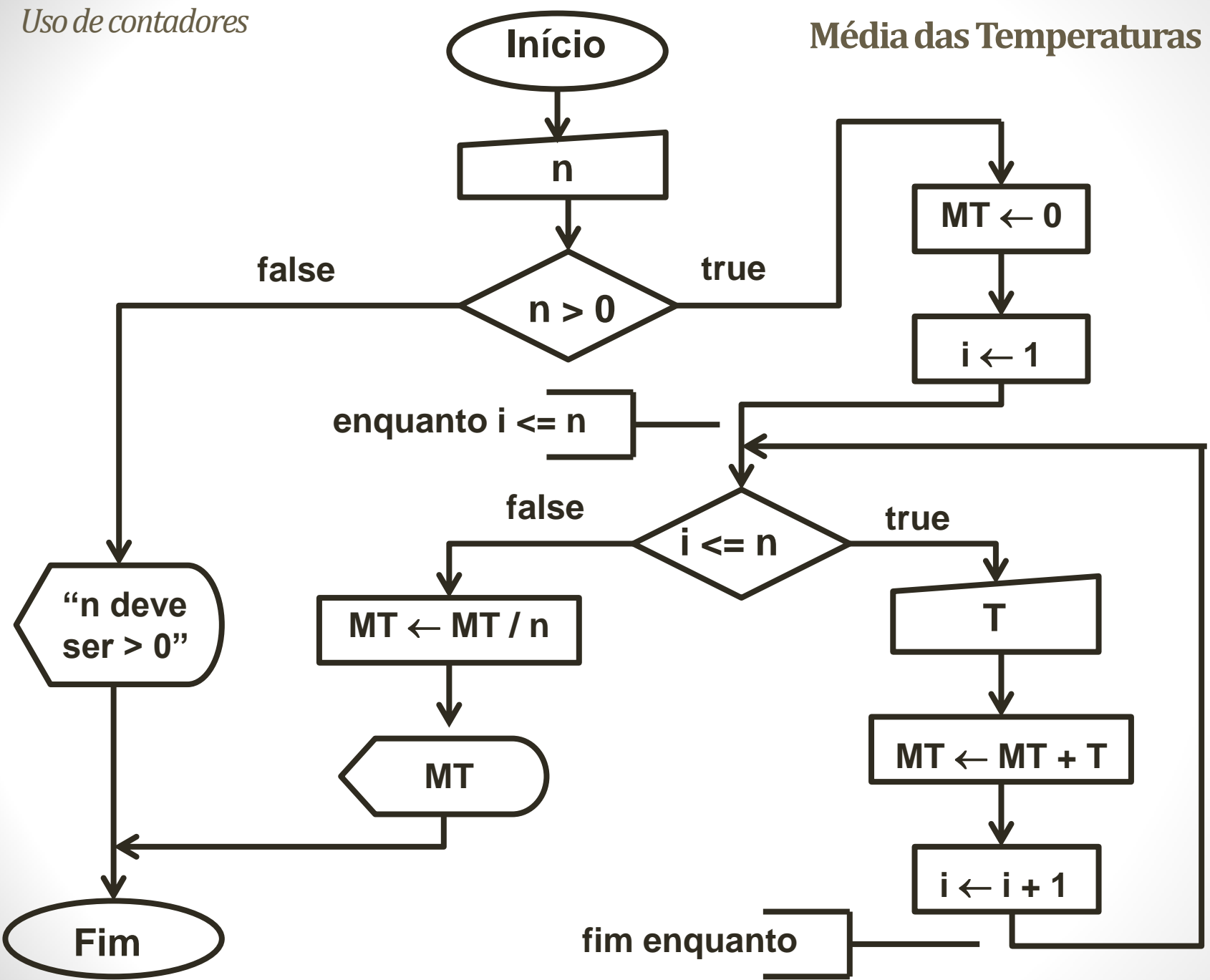


Exercício: Média das Temperaturas

- Durante uma semana, foram medidas n temperaturas ambiente, no campus da UFOP, às 22:00 h.
- Projete um fluxograma que tenha como entrada as n temperaturas, e como saída a temperatura média referente a essas temperaturas.
- **Observações:**
 1. O número de temperaturas é variável, mas deve-se tratar o caso de $n \leq 0$, pois pode-se ocasionar a divisão por zero na média;
 2. Se n pode assumir qualquer valor positivo, serão lidas n temperaturas diferentes, quantas variáveis são necessárias?

Exercício: Média das Temperaturas

- **Resposta para OBS 2:** Somente uma!
 - Um comando de repetição ficará encarregado de ler cada temperatura e acumular seu valor em uma soma.
 - Ao final, as n temperaturas estarão somadas, faltando apenas calcular a média;
 - As somas serão realizadas em uma variável denominada **variável acumuladora**;
 - A técnica é inicializar a variável acumuladora com zero (elemento neutro da adição) fora do laço;
 - Desta forma, a cada leitura de uma nova temperatura, dentro da repetição, soma-se a temperatura corrente à variável acumuladora;
- A seguir, o fluxograma.



Tipos de dados;
Uso de contadores;
Comandos de repetição/iteração;
Exercícios.

COMANDOS DE REPETIÇÃO

Introdução

- Para permitir que uma operação seja executada repetidas vezes utiliza-se **comandos de repetição**;
- Uma estrutura deste tipo também é chamada de **laço** (do inglês **loop**);
- No Scilab, são definidos dois comandos de repetição:
 1. Laço controlado por contador (**for**);
 2. Laço controlado logicamente (**while**).

Introdução

- Em um **laço controlado por contador**, os comandos são repetidos um número **predeterminado** de vezes;
- Já em um **laço controlado logicamente**, os comandos internos (corpo do laço) são repetidos indefinidamente **enquanto** uma expressão lógica for verdadeira;
- Denomina-se **iteração** a repetição de um conjunto de comandos;
 - Cada execução do corpo do laço, juntamente com a condição de terminação do laço, é uma iteração.

Laço controlado por contador

- O comando **for** é um laço controlado por contador, e é definido da seguinte forma:

```
for variável = <valor inicial> : <valor final>  
    <conjunto de comandos>  
end
```

- <conjunto de comandos> é o conjunto de instruções a serem executadas, é denominado corpo do laço;
- **variável = <valor inicial> : <valor final>** é a declaração da variável contadora em conjunto com a definição dos valores inicial e final do laço, a cada iteração a variável será incrementada de 1;
- **for** e **end** são palavras reservadas da linguagem.

Fatorial

- Considere o problema do cálculo do fatorial;
- O fatorial de um número N (**N!**) é dado por:

$$N! = 1 * 2 * 3 * \dots * (N-1) * N$$

- Sendo que o fatorial de **0** é **1**, por definição;
- Embora exista uma função no Scilab que retorna o fatorial de um número (***factorial(n)***), estamos interessados agora na lógica por trás deste problema: vejamos um programa que resolve o problema.

Fatorial

```
n = input("Entre com um numero");  
fat = 1;  
for cont = 2:n  
    fat = fat * cont;  
end  
printf("Fatorial de %g e igual a %g\n", ...  
    n, fat);
```

Somatório 1

- Elabore um programa que calcule e imprima o valor de S:

$$S = \frac{1}{1} + \frac{3}{2} + \frac{5}{3} + \frac{7}{4} + \dots + \frac{99}{50}$$

- Dica, encontre o **padrão** entre o numerador e o denominador.

Somatório 1

- Elabore um programa que calcule e imprima o valor de S:

$$S = \frac{1}{1} + \frac{3}{2} + \frac{5}{3} + \frac{7}{4} + \dots + \frac{99}{50}$$

- Dica, encontre o **padrão** entre o numerador e o denominador:
 - **Numerador = 2 * Denominador - 1.**

Somatório 1

```
s = 0;  
for d = 1:50  
    s = s + (2 * d - 1) / d;  
end  
printf("Valor de S = %g\n", s);
```

Somatório 2

- Agora vamos mudar o problema anterior para:

$$S = \frac{1}{1} + \frac{5}{3} + \dots + \frac{97}{49}$$

- O padrão entre o numerador e o denominador é o mesmo, mas agora o denominador varia de forma diferente.

Somatório 2

```
s = 0;
for d = 1:50
    if (modulo(d, 2) == 1) then
        s = s + (2 * d - 1) / d;
    end
end
printf("Valor de S = %g\n", s);
```

Somatório 2

Ou então:

```
s = 0;
```

```
for d = 1:2:50
```

```
    s = s + (2 * d - 1) / d;
```

```
end
```

```
printf("Valor de S = %g\n", s);
```

Realiza um incremento de 2
na variável *i* a cada iteração.

Nova sintaxe para o **for**

- Agora o comando **for** pode ser definido da seguinte forma:

```
for variável = <valor inicial> : <passo> : <valor final>  
    <conjunto de comandos>  
end
```

- **<conjunto de comandos>** é o conjunto de instruções a serem executadas, é denominado corpo do laço;
- **variável = <valor inicial> : <passo> : <valor final>** é a declaração da variável contadora em conjunto com a definição dos valores inicial, final e o **passo** do laço, a cada iteração a variável será incrementada pelo valor do **passo**;
- **for** e **end** são palavras reservadas da linguagem.

Variável contadora

- Os valores assumidos pela variável contadora não precisam ser inteiros, por exemplo:

```
for x = 0 : 0.3 : 0.7
    printf("\nX = %g", x);
end
```

- Este programa é válido, e resultará em:

```
X = 0
X = 0.3
X = 0.6
```

Tabela de senos

- Elabore um programa que calcule e imprima uma tabela de senos, conforme a tabela abaixo:

x	seno(x)
0.0	0.0000
0.2	0.1987
0.4	0.3894
0.6	0.5646
0.8	0.7174

- O critério de parada é $x = 2\pi$.

Tabela de senos

- **Solução:**

```
printf("\n x seno(x)");
for x = 0 : 0.2 : 2 * %pi
    printf("\n %3.1f %7.4f", x, sin(x));
end
```

- **Saída:**

x	seno(x)
0.0	0.0000
0.2	0.1987
0.4	0.3894
0.6	0.5646
:	

Tabela de senos

- **Observações:**
 - Perceba que os valores da variável contadora podem ser definidos por expressões (**2 * %pi**);
 - É possível formatar a saída dos valores no *printf* para obter uma tabela:
 - Não existe somente o **%g**;
 - Neste exemplo:
 - **%3.1f** indica um valor *float* (número fracionário) com um total de 3 caracteres, com 1 casa decimal;
 - **%7.4f** indica um valor *float* com um total de 7 caracteres, com quatro casas decimais.

Somatório 3

- Agora vamos mudar novamente o problema do somatório:

$$S = \frac{97}{49} + \dots + \frac{5}{3} + \frac{1}{1}$$

- Agora houve uma inversão na sequência dos termos, o que fazer?

Somatório 3

Realiza um decremento de 2 na variável *i* a cada iteração.

```

s = 0;
for d = 49:-2:1
    s = s + (2 * d - 1) / d;
end
printf("Valor de S = %g\n", s);
  
```

Laço controlado logicamente

- O comando **while** é um laço controlado logicamente;
- O laço **while** é definido da seguinte forma:

```
while <expressão lógica>  
    <conjunto de comandos>  
end
```

- <conjunto de comandos> é o conjunto de instruções a serem executadas, é denominado corpo do laço;
- <expressão lógica> é a expressão que define quando os comandos deverão ser executados;
- **while** e **end** são palavras reservadas da linguagem.

Equivalência entre **while** e **for**

- Todo comando **for** pode ser substituído por um comando **while**, por exemplo:

```

for x = 0 : 0.2 : 2 * %pi
    printf("\n %3.1f %7.4f", x, sin(x));
end
  
```

- Pode ser escrito como:

```

x = 0;
while x <= 2 * %pi
    printf("\n %3.1f %7.4f", x, sin(x));
    x = x + 0.2;
end
  
```

Equivalência entre **while** e **for**

- No exemplo anterior, o uso do **for** é mais adequado;
- Mas, existem situações em que o comando **while** é mais adequado, ou, que não será possível utilizar o comando **for**;
- A seguir, dois exemplos.

Equivalência entre **while** e **for**

- Validação de dados de entrada:

```
a = input ("Entre com o valor de a: ");  
while (a == 0)  
    printf ("a não pode ser 0.\n");  
    a = input ("Entre com o valor de a: ");  
end
```

- Não é possível “prever” quantas vezes o usuário entrará com um valor incorreto;
- Não é possível utilizar o **for** neste caso.

Equivalência entre **while** e **for**

- Implementando o Algoritmo de Euclides para obter o MDC:

```
x = input("x = ");
```

```
y = input("y = ");
```

```
xa = x;
```

```
ya = y;
```

```
while y <> 0
```

```
    r = modulo(y, x);
```

```
    x = y;
```

```
    y = r;
```

```
end
```

```
printf("mdc(%d,%d) = %d", xa, ya, x)
```

Mais uma vez, não é possível “prever” os valores da variável contadora para a utilização do comando **for**.

Equivalência entre **while** e **for**

- **Observações:**

- Use o **for** sempre que possível, ele será mais **seguro** e **eficiente**;
- Cuidado ao utilizar o **while**, pois será possível que o loop nunca termine (**loop infinito**), exemplos:

```
x = 0;
while x <= 10
    printf("\nx = %g", x)
end
```

O valor de x nunca será alterado. Com isso, nunca deixará o loop.

```
x = 0;
while x <= 10
    printf("\nx = %g", x)
    x = x - 0.2;
end
```

O valor de x é iniciado com 0 e depois é decrementado dentro do loop. Com isso, nunca deixará o loop.

Outro exemplo de **while**

- Para repetir a execução do programa enquanto o usuário assim desejar:

```

continua = %t;
while continua
    // Comandos do seu programa
    :
    :
    // Decisão sobre a continuação do programa
    decisao = input("Continuar? (s/n)", "string");
    continua = decisao == "s" | decisao == "S";
end
printf ("Término do programa.\n");
  
```

Laços aninhados

- Considere o programa:

```
for j = 1:4  
    printf("x");  
end
```

- Como resultado teremos:

```
xxxx
```

- E se agora eu desejar imprimir um número arbitrário de linhas com 4 caracteres “x”?

Laços aninhados

- Dentro de um bloco de comandos pode haver qualquer outro comando;
- Assim, dentro de um for pode haver outro for;
- Resolvendo o problema:

```
lin = input("Numero de linhas: ");  
for i = 1 : lin  
    for j = 1 : 4  
        printf("x");  
    end  
    printf("\n");    // mudança da linha  
end
```

- **Exercício:** E se agora eu desejar também um número arbitrário de colunas?

Tabuada de Multiplicação

- **Exercício:** Faça um programa que imprima a tabela da tabuada de multiplicação:

	1	2	3	4	5	6	7	8	9	10
1	1	2	3	4	5	6	7	8	9	10
2	2	4	6	8	10	12	14	16	18	20
3	3	6	9	12	15	18	21	24	27	30
4	4	8	12	16	20	24	28	32	36	40
5	5	10	15	20	25	30	35	40	45	50
6	6	12	18	24	30	36	42	48	54	60
7	7	14	21	28	35	42	49	56	63	70
8	8	16	24	32	40	48	56	64	72	80
9	9	18	27	36	45	54	63	72	81	90
10	10	20	30	40	50	60	70	80	90	100

Tabuada de Multiplicação

- Solução:

```
clc;
printf("\nTabuada de Multiplicação:\n\n");
printf("  | 1  2  3  4  5  6  7  8  9  10\n");
printf("-----\n");
for linha = 1 : 10
    printf("%2.0f |", linha);
    for coluna = 1 : 10
        printf("%3.0f ", linha * coluna);
    end
    printf("\n");
end
```

Tabuada de Multiplicação

- Saída:

Tabuada de Multiplicação:

		1	2	3	4	5	6	7	8	9	10

1		1	2	3	4	5	6	7	8	9	10
2		2	4	6	8	10	12	14	16	18	20
3		3	6	9	12	15	18	21	24	27	30
4		4	8	12	16	20	24	28	32	36	40
5		5	10	15	20	25	30	35	40	45	50
6		6	12	18	24	30	36	42	48	54	60
7		7	14	21	28	35	42	49	56	63	70
8		8	16	24	32	40	48	56	64	72	80
9		9	18	27	36	45	54	63	72	81	90
10		10	20	30	40	50	60	70	80	90	100

Tipos de dados;
Uso de contadores;
Comandos de repetição/iteração;

Exercícios.

EXERCÍCIOS

Pagando a Conta

- Um aluno foi ao supermercado e gastou X reais com as compras da semana.
- Escrevera um programa que tenha como entrada o valor X da compra. O programa deve determinar quantas notas de 50, de 10 e de 1 real são suficientes para o pagamento da compra.
- **Obs:** O programa só deverá imprimir a quantidade de notas que forem maiores do que zero.

Pagando a Conta

```
clc;  
ValorCompra = input("VALOR DA COMPRA: ");  
N50 = 0; N10 = 0;  
while (ValorCompra >= 50)  
    ValorCompra = ValorCompra - 50;  
    N50 = N50 + 1;  
end  
while (ValorCompra >= 10)  
    ValorCompra = ValorCompra - 10;  
    N10 = N10 + 1;  
end
```

Pagando a Conta

```
printf("O VALOR DA COMPRA SERÁ PAGO COM:\n");  
if (N50 > 0) then  
    printf("%g NOTA(S) DE CINQUENTA\n", N50);  
end  
if (N10 > 0) then  
    printf("%g NOTA(S) DE DEZ\n", N10);  
end  
if (ValorCompra > 0) then  
    printf("%g NOTA(S) DE UM\n", ValorCompra);  
end
```

Decimal para Binário

- Escreva um programa que tenha como entrada um valor na base 10;
- O programa gerará o valor correspondente na base 2, ou seja, o equivalente do número decimal em binário.

Decimal para Binário

```
numero = input("DIGITE UM DECIMAL: ");
printf("O EQUIVALENTE EM BINÁRIO É:\n");
printf("OBS: LEIA O BINÁRIO DA ");
printf("DIREITA PARA A ESQUERDA\n\n");
quociente = int(numero / 2);
while (quociente <> 0)
    digito = modulo(numero, 2);
    printf("%g", digito);
    numero = quociente;
    quociente = int(numero / 2);
end
digito = modulo(numero, 2);
printf("%g", digito);
```

Lista 3 do prof. David

- Resolução dos exercícios da lista conforme distribuição predefinida.

Próxima aula prática: resolução de exercícios com o Scilab.

Próxima aula teórica: Variáveis Homogêneas - Vetores..

FIM!

DÚVIDAS?