Programando com Scilab

(Versão deste texto: 0.27)

E. G. M. de Lacerda
Departamento de Engenharia de Computação e Automação (DCA)
UFRN
estefane@dca.ufrn.br

13 de Junho de 2011

Resumo

Este curso apresenta a linguagem de programação do Scilab da mesma forma que um curso tradicional de introdução a programação. O Scilab é um ambiente de programação dedicado a resolução de problemas científicos e de engenharia. Ele está disponível[1] para vários sistemas operacionais tais como Windows, Linux e Mac OS X. O material deste curso pode ser usado em qualquer um desses sistemas. Procure seu professor ou seu centro de computação local para informações sobre como o Scilab está instalado localmente.

[1]Página do Scilab: `http://www.scilab.org`

Sumário

1 Preliminares **1**
- 1.1 Usando o Console do Scilab como uma Simples Calculadora 1
- 1.2 Variáveis e o Comando de Atribuição 2
  - 1.2.1 Regras para Formação de Nomes de Variáveis 3
  - 1.2.2 O Ponto e Vírgula 3
- 1.3 Expressões Aritméticas 3
  - 1.3.1 Funções Matemáticas Comuns 4
  - 1.3.2 Funções de Arredondamento 4
  - 1.3.3 Ordem de Avaliação entre Operadores Aritméticos 5
- 1.4 Strings 6
- 1.5 Números Complexos 8
- 1.6 O Espaço de Trabalho 9
  - 1.6.1 O Comando Clear 9
  - 1.6.2 Os Comandos Save e Load 9
- 1.7 Formato de Visualização dos Números 10
- 1.8 Constantes Especiais do Scilab 11
- 1.9 A Variável ans 11
- 1.10 Ajuda 11
- 1.11 Exercícios 11

2 Arquivos de Scripts **13**
- 2.1 Comando de Entrada de Dados 13
- 2.2 Comandos de Saída de Dados 13
- 2.3 Arquivos de Scripts 15
- 2.4 Criando Arquivos de Script 16
- 2.5 Executando Arquivos de Script 17
- 2.6 Exemplos 18
- 2.7 Linhas de Comentários 19
- 2.8 Alterando o Diretório de Trabalho 19

3 Estruturas de Seleção **21**
- 3.1 Estruturas de Controle 21
- 3.2 Expressões Booleanas 22
- 3.3 Variáveis Booleanas 23
- 3.4 Tipos de Dados Primitivos 23
- 3.5 Ordem de Avaliação entre os Operadores 23
- 3.6 A Seleção Simples IF-END 24
- 3.7 A Seleção Bidirecional IF-ELSE-END 25
- 3.8 Aninhando Seletores 26

4 Estruturas de Repetição **32**
4.1 Laços 32
4.2 Laço Controlado Logicamente 32
4.3 Laço Controlado por Contador 35
4.4 Exemplos com Laços 36
4.5 Laços Aninhados 39

5 Matrizes **42**
5.1 Vetores 42
5.1.1 Acessando Elementos do Vetor 43
5.2 Matrizes Bidimensionais 45
5.3 Vetores de String 48
5.4 Estudo de Caso 49
5.5 Exemplos com Matrizes 52
5.5.1 Ordenação de Vetores 52
5.5.2 Gerando Números Aleatórios 54
5.5.3 Uma Aplicação de Matrizes 55

6 Manipulação Matricial **58**
6.1 Construção de Matrizes 58
6.2 Secionamento de Matrizes 61
6.2.1 Indexação Linear 64
6.3 O Operador $ 65
6.4 Atribuição 66
6.5 Dimensão de Matrizes 67
6.6 Operações Escalar-Matriz 68
6.7 Operações Matriz-Matriz 69
6.8 Solução de Sistemas de Equações Lineares 72
6.9 Transposta de Matrizes Complexas 72
6.10 Zeros e Ones 73

7 Funções **74**
7.1 Introdução 74
7.2 Parâmetros de Entrada e Saída 74
7.3 Funções Definidas pelo Usuário 75
7.4 A Idéia Básica das Funções 76
7.5 Escopo de Variáveis 78
7.5.1 Variáveis Locais 78
7.5.2 Variáveis Globais 79
7.6 Os Programas do Scilab 80
7.7 Passagem de Parâmetros 81
7.8 Exemplos 81
7.9 O Comando return 83
7.10 Estudo de Caso: Um Programa de Estatística 84
7.10.1 O Comando de Múltipla Escolha SELECT-CASE 86

Capítulo 1

PRELIMINARES

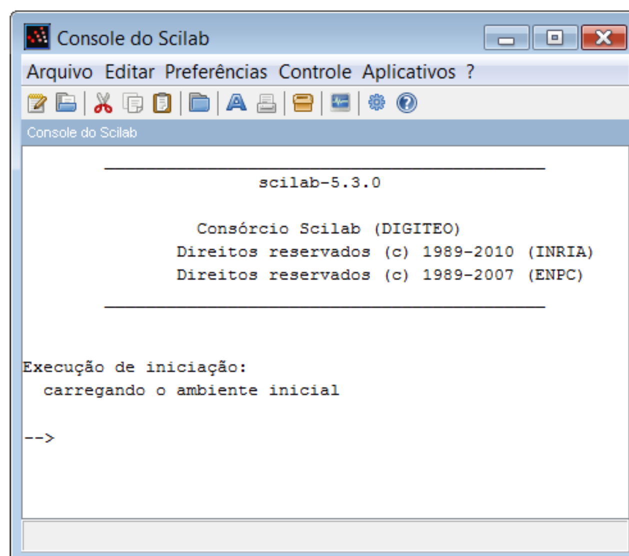
NESTE capítulo serão apresentados o ambiente de trabalho do Scilab e conceitos básicos de programação como variáveis, tipos de dados e expressões aritméticas.

1.1 USANDO O CONSOLE DO SCILAB COMO UMA SIMPLES CALCULADORA

O Scilab pode fazer operações aritméticas com números reais e complexos. Os operadores aritméticos são:

Adição	+
Subtração	-
Multiplicação	*
Divisão	/
Potenciação	^

Quando o Scilab é executado surge uma janela chamada de **console** do Scilab como mostrado na Figura a seguir. O console do Scilab é o mecanismo usado pelo usuário para interagir com o Scilab por meio de comandos[1].



[1]Em computação, esta forma de interagir com o computador é chamada de Interface de Linha de Comandos. Consiste em digitar uma linha de texto representando um comando ou tarefa a ser executada pelo computador. É parte fundamental de muitos softwares científicos (e.g., Matlab, Maple) e está presente em várias linguagens de programação (e.g., Python e Perl) e sistemas operacionais como Linux (Shell) e Windows (DOS e Powershell).

Procure pelo símbolo:

```
-->
```

Ele é chamado de *prompt* e indica que o console do Scilab está esperando um comando a ser digitado pelo usuário. Portanto, as operações aritméticas são digitados após o símbolo -> e em seguida tecla-se [ENTER]. Exemplo:

```
-->2+3 [ENTER]
 ans  =

    5.
```

Outros exemplos:

```
-->5+6/2
 ans  =

    8.

-->4^2                  // 4 elevado a potência de 2
 ans  =

    16.
```

1.2 VARIÁVEIS E O COMANDO DE ATRIBUIÇÃO

Uma variável é uma abstração de uma célula ou um conjunto de células na memória do computador. Informações são armazenas em variáveis para posterior uso. Muitos programadores costumam a interpretar variáveis como sendo um nome para uma posição na memória do computador.

Fundamental na programação, o comando de atribuição é usado para atribuir ou modificar a informação contida na variável. No Scilab, usa-se o símbolo = para atribuição. Exemplo: digite estes comandos no console:

O símbolo de atribuição = não significa igualdade matemática, uma vez que o comando de atribuição `i = i+1` é válido, mas não representa igualdade matemática.

```
-->a = 2                        //  Atribui 2 para variável a
 a  =

    2.

-->b = 4                        //  Atribui 4 para variável b
 a  =

    4.

-->area = a*b                   //  Atribui o produto de a e b
 area  =                        //  para variável area

    8.

-->b = b+1                      //  Incrementa uma unidade
 b  =                           //  na variável b

    5.
```

1.2.1 Regras para Formação de Nomes de Variáveis

Os nomes de variáveis (também conhecidos por *identificadores*) devem seguir as seguintes regras:

1. Nomes de variáveis começam com uma letra seguido de letras, algarismos ou sublinhados. Por exemplo: `Alpha`, `notas`, `A1`, `B23` e `cor_do_objeto`;
2. Caracteres especiais não são permitidos. Isto é, não é permitido usar `#`, `$`, `&`, `%`, `?`, `!`, `@`, <, `~`, etc;
3. Caracteres acentuados não são permitidos;
4. Nomes de variáveis são sensíveis a maiúsculas e minúsculas. Por exemplo, variável `Alpha` é diferente das variáveis `ALPHA`, `alpha` e `AlPhA`.

De acordo com as regras acima, os seguintes nomes de variáveis são válidos:

`ALPHA`, `X`, `B1`, `B2`, `b1`, `matricula` e `MEDIA`.

Porém, estes nomes de variáveis são inválidos:

`5B`, `1b`, `nota[1]`, `A/B` e `X@Z`.

1.2.2 O Ponto e Vírgula

A ausência ou presença do ponto e vírgula no final de um comando do Scilab visualiza ou suprime, respectivamente, o resultado do cálculo. Por exemplo, o resultado do seguinte comando, digitado *com* ponto e vírgula, é suprimido:

```
-->A = 4+4^2;

-->
```

Se este comando é digitado sem ponto e vírgula, o resultado é visualizado:

```
-->A = 4+4^2
 A  =

    20.
```

Mais exemplos:

```
-->a=2;

-->b=4;

-->area=a*b                 // aqui o ponto e vírgula foi
 area  =                    // suprimido porque precisamos
                            // visualizar o resultado.
    8.
```

1.3 EXPRESSÕES ARITMÉTICAS

Os operadores aritméticos combinam números e variáveis para formar expressões aritméticas. Exemplos:

```
A+B*C
(NOTA1+NOTA2)/2
1/(a^2+b^2)
```

Além dos operadores aritméticos podemos usar funções matemáticas. Exemplos:

```
2+3*cos(x)
X^(2*sin(y))
2+3*tan(x)+K^2
```

1.3.1 Funções Matemáticas Comuns

As principais funções matemáticas do Scilab são mostradas na Tabela 1.1. O número π e a base do logaritmo natural $e = 2,718281828...$ são representadas pelas variáveis especiais `%pi` e `%e`, respectivamente. Exemplos:

```
-->cos(2*%pi)                // coseno de 2 vezes PI
 ans  =

    1.

-->%e^2                      // 2,718281828 ao quadrado
 ans  =

    7.389056098931

-->abs(-5)                   // valor absoluto
 ans  =

    5.

-->modulo(8,3)               // Resto da divisão entre 8 e 3
 ans  =

    2.

-->modulo(6,3)
 ans  =

    0.

-->sign(-4)
 ans  =

  - 1.

-->sign(5)
 ans  =

    1.
```

1.3.2 Funções de Arredondamento

As principais funções de arredondamento do Scilab são mostradas na Tabela 1.2. Exemplos:

```
-->a = 34.885;

-->fix(a)             // trunca a parte fracionária
 ans  =
```

Tabela 1.1: Funções Matemáticas

`abs(x)`	Valor absoluto.
`acos(x)`	Arco co-seno.
`acosh(x)`	Arco co-seno hiperbólico.
`asin(x)`	Arco seno.
`asinh(x)`	Arco seno hiperbólico.
`atan(x)`	Arco tangente.
`atanh(x)`	Arco tangente hiperbólico.
`conj(x)`	Conjugado.
`cos(x)`	Co-seno.
`cosh(x)`	Co-seno hiperbólico.
`exp(x)`	Exponencial: e^x.
`imag(x)`	Parte imaginária de um número complexo.
`log(x)`	Logaritmo natural.
`log10(x)`	Logaritmo na base 10.
`real(x)`	Parte real de um número complexo.
`modulo(x,y)`	Resto da divisão de x por y.
`sign(x)`	Função sinal: retorna o valor -1, +1 ou zero conforme o argumento x seja negativo, positivo ou nulo, respectivamente.
`sin(x)`	Seno.
`sinh(x)`	Seno hiperbólico.
`sqrt(x)`	Raiz quadrada.
`tan(x)`	Tangente.
`tanh(x)`	Tangente hiperbólica.

Tabela 1.2: Funções de Arredondamento

`ceil(x)`	Arredondamento na direção de mais infinito ($+\infty$).
`fix(x)`	Arredondamento na direção de zero (isto é, devolve a parte inteira de um número).
`floor(x)`	Arredondamento na direção de menos infinito ($-\infty$).
`int(x)`	Mesmo que `fix`.
`round(x)`	Arredondamento para o inteiro mais próximo.

```
    34.

-->round(a)          // arredonda para o inteiro mais próximo
 ans  =

    35.

-->ceil(3.1)          // arredonda para mais.
 ans  =

    4.
```

Outros exemplos são mostrados na Tabela 1.3.

1.3.3 Ordem de Avaliação entre Operadores Aritméticos

Para uma boa avaliação de uma expressão aritmética é necessário se familiarizar com a *ordem de avaliação dos operadores*. Aqui, as regras importantes são: prioridade, associatividade e parênteses.

Tabela 1.3: Exemplos de Funções de Arredondamento

x	ceil(x)	floor(x)	fix(x)	round(x)
1.75	2	1	1	2
1.5	2	1	1	2
1.25	2	1	1	1
-1.25	-1	-2	-1	-1
-1.5	-1	-2	-1	-2
-1.75	-1	-2	-1	-2

Operações de alta prioridade são realizadas primeiro do que as operações de baixa prioridade. A ordem de prioridade entre os operadores é a seguinte:

Prioridade	Operação	Associatividade
1ª	Potenciação	da direita para a esquerda
2ª	multiplicação, divisão	da esquerda para a direita
3ª	Adição, subtração	da esquerda para a direita

Exemplos:

`2+10/5` ⟵`10/5` é avaliada primeiro.

`A+B/C+D` ⟵`B/C` é avaliada primeiro.

`R*3+B^3/2+1` ⟵`B^3` é avaliada primeiro.

Associatividade é a regra usada quando os operadores têm a mesma prioridade. Por exemplo, para as operações de adição e subtração (que possuem mesma prioridade) a regra de associatividade diz que a operação mais a esquerda é avaliada primeiro:

`A-B+C+D` ⟵`A-B` é avaliada primeiro, porque está mais a esquerda.

O mesmo vale para multiplicação e divisão:

`A*B/C*D` ⟵`A*B` é avaliada primeiro, porque está mais a esquerda.

No entanto, para potenciação, a regra da associatividade diz que a operação mais a direita deve ser avaliada primeiro:

`A^B^C^D` ⟵`C^D` é avaliada primeiro, porque está mais a direita.

A ordem de prioridade pode ser alterada pelo uso do parênteses:

`(A+4)/3` ⟵`A+4` é avaliada primeiro devido aos parênteses.

`(A-B)/(C+D)` ⟵`A-B` é avaliada primeiro. Depois a adição. Por último, a divisão.

`R*3+B^(3/2)+1` ⟵`3/2` é avaliada primeiro.

1.4 STRINGS

Strings são usados para toda e qualquer informação composta de caracteres alfanuméricos e/ou caracteres especiais (exemplo, `#`, `$`, `&`, `%`, `?`, `!`, `@`, `<`, `~`, etc). Os strings são envolvidos

por aspas duplas ou simples. Exemplos:[2].

```
-->a = "abcd"
 a  =

 abcd

-->b = 'efgh'
 b  =

 efgh

-->c = "Maria e Jose"
 c  =

 Maria e Jose
```

Um das atividades mais comuns em programação é a concatenação de strings. Concatenação é a junção de dois ou mais strings. Isto pode ser feito com o operador +.

```
-->a + b               // Concatena abcd com efgh
 ans  =

 abcdefgh

-->n = "Pedro"
 n  =

 Pedro

-->m = "Paulo"
 m  =

 Paulo

-->m + n               // Concatena Paulo com Pedro sem
 ans  =                // espaço entre eles.

 PauloPedro

-->m + " e " + n       // Concatena Paulo com Pedro
 ans  =                // inserindo espaços entre eles.

 Paulo e Pedro
```

Muitas vezes precisamos armazenar informações que contém as aspas. Isto pode ser feito repetindo as aspas. Exemplos:

```
-->n = "O oráculo disse ""conheça-te a ti mesmo"" para Socrátes."
 n  =

 O oráculo disse "conheça-te a ti mesmo" para Socrátes.
```

Algumas funções para manipulação de strings são mostradas da Tabela 1.4. Exemplos:

[2]Devemos usar aspas duplas ou simples? A aspa simples é também usado como operador de transposta hermitiana (mas isto não acarreta problemas de programação). Por isso é melhor usar as aspas duplas que não possui tal duplo sentido e torna seu programa um pouco mais legível.

Tabela 1.4: Funções de Manipulação de String

`convstr`	Retorna os caracteres de um string convertidos para maiúscula ou minúscula.
`length`	Comprimento de um string.
`part`	Extrai caracteres de um string.
`strindex`	Procura a posição de um string dentro de outro.
`strcat`	Concatena strings.
`string`	Converte número para string.
`evstr`	Converte string em número. Também avalia expressões aritméticas.
`eval`	Converte string em número. Também avalia expressões aritméticas.
`strsubst`	Substitui uma parte de um string por um outro string.

```
-->m = "Pedro";

-->length(m)           // Comprimento do string "Pedro"
 ans  =

    5.
```

Para concatenar números com strings use a função `string()`.

```
-->a = "a camisa " + string(10)
 a  =

 a camisa 10
```

Para somar uma string com um número use `evstr()`:

```
-->a = "12" + "34"
 a  =

 1234

-->evstr(a) + 10
 ans  =

    1244.
```

1.5 NÚMEROS COMPLEXOS

Não é necessário manuseio especial em Scilab para números complexos. As operações com números complexos são tão fáceis como nos reais. A unidade imaginária é representado por `%i`, ou seja, `%i` é igual a `sqrt(-1)`. Exemplos:

```
x = 3 + 4*%i
y = 1 - %i
z1 = x - y
z2 = x * y
z3 = x / y
real(z1)                      ←—Parte real de z1
imag(z1)                      ←—Parte imaginária de z1
abs(x)                        ←—Valor absoluto do número complexo
atan(imag(x),real(x))         ←—Argumento do número complexo
conj(z2)                      ←—Conjugado
sin(x)                        ←—Seno de um número complexo
```

1.6 O ESPAÇO DE TRABALHO

Quando um comando de atribuição como este:

```
-->x = 3
```

é digitado no Scilab, a variável `x` é armazenada em uma área da memória do Scilab denominada de **Espaço de Trabalho** (do inglês, *workplace*). O Espaço de Trabalho é uma parte da memória do computador que armazena as variáveis criadas pelo console do Scilab e pelos arquivos de Script (mostrados adiante).

1.6.1 O Comando Clear

O comando `clear` apaga todas a variáveis do Espaço de Trabalho criadas pelo usuário. Exemplo:

```
-->clear               // Apaga todas a variáveis
```

O comando `clear` seguido de nome de uma variável apaga somente a variável:

```
-->a = 2;

-->b = 3;

-->c = 4;

-->clear b;            // Apaga somente b deixando as
                       // outras variáveis intactas.
```

O comando `who` mostra todas as variáveis do Espaço de Trabalho.

1.6.2 Os Comandos Save e Load

As variáveis são apagadas quando o usuário termina a execução do Scilab. Para usá-las da próxima vez que executar o Scilab, você deve salva-las com o comando `save(arquivo)`. Por exemplo,

```
-->a = 2;

-->b = 3;

-->c = 4;

-->save("dados.dat");
```

As variáveis foram salvas no arquivo `dados.dat`. O comando `load(arquivo)` é usado para recuperar variáveis que foram salvas no arquivo. Por exemplo,

```
-->clear                   // apaga todas as variáveis

-->a+b                     // variáveis a e b não existem
  !--error     4           // porque foram apagadas
undefined variable : a


-->load("dados.dat");      // recupera as variáveis a, b e c
```

```
-->a+b                     // Ok!
 ans  =

    5.
```

1.7 FORMATO DE VISUALIZAÇÃO DOS NÚMEROS

O comando `format` modifica a quantidade de dígitos com que os números são mostrados no Scilab. Por exemplo, o comando

```
--> format(5)
```

fará com que todas os números sejam visualizados em 5 posições (incluindo o ponto decimal e um espaço para o sinal). Por exemplo,

```
-->sqrt(3)
 ans  =

    1.73
```

Para aumentar o números de posições para 16, usa-se

```
-->format(16)

-->sqrt(3)
 ans  =

    1.7320508075689
```

A raiz de 3 foi mostrada ocupando 16 posições (sendo uma posição para o ponto, um espaço reservado para o sinal, uma posição para a parte inteira e 13 posições para a parte fracionária).

O comando `format('e')` mostra os números em **notação científica**. Por exemplo,

```
-->format('e')

-->2*%pi/10
 ans  =

    6.283185307E-01
```

`6.283185307E-01` significa $6.283185307 \times 10^{-1}$. Para retornar ao formato inicial usa-se,

```
-->format('v')
```

que é chamado de "formato de variável". Vejamos outras formas de usar o comando `format`:

```
-->format('v',10)
```

mostra os números em formato de variável com 10 posições.

```
-->format('e',8)
```

mostra os números em notação científica com 8 posições.

1.8 CONSTANTES ESPECIAIS DO SCILAB

O Scilab possue várias constantes pré-definidas. Algumas constantes pré-definidas não podem ser alteradas.

`%pi`	O número π.
`%eps`	Constante que representa a precisão numérica da máquina. É o menor número que, somado a 1, resulta em um número maior que 1 no computador.
`%inf`	Representa infinito ∞.
`%nan`	Não numérico (do inglês, *Not A Number*).
`%i`	$\sqrt{-1}$.
`%e`	A base do logaritmo natural.
`%t` ou `%T`	Representa o valor booleano verdadeiro.
`%f` ou `%F`	Representa o valor booleano falso.
`%s`	Um polinômio com uma única raiz em zero e `s` como o nome da variável. A constante `%s` é definida como `poly(0,'s')`.
`%z`	Um polinômio com uma única raiz em zero e `s` como o nome da variável. A constante `%z` é definida como `poly(0,'z')`.

1.9 A VARIÁVEL ANS

A variável `ans` (abreviação da palavra inglesa *answer*) armazena o valor corrente de saída do Scilab. Pode-se usar `ans` para efetuar cálculos porque ela armazena o valor do último cálculo realizado. Exemplo:

```
-->4+5
 ans  =

    9.

-->cos(ans)+3
 ans  =

    2.0888697
```

1.10 AJUDA

O comando `help` informa sobre comandos e funções do Scilab. Por exemplo:

`help cos`	⟵Informa sobre a função que calcula o co-seno
`help ceil`	⟵Informa sobre a função `ceil`

O comando `apropos` procura comandos e funções utilizando uma palavra-chave. Por exemplo, se não sabemos o nome da função que calcula o seno hiperbólico, podemos digitar algo como

```
--> apropos hyperbolic
```

e o Scilab mostrará todas a funções relacionadas com a palavra-chave *hyperbolic*.

1.11 EXERCÍCIOS

1. O que são variáveis?

2. Quais os tipos primitivos de informação manipuladas pelo Scilab?

3. Assinale os identificadores (nomes de variáveis) válidos:

a) `(X)`	b) `XMU2`	c) `AH!`	d) `NOTA1`
e) `5NOTA`	f) `“NOTA1”`	g) `A[4]`	h) `A&B`
i) `A+B`	j) `I00001`	l) `NOTA/2`	m) `PEDROEPAULO`

4. Escreva as declarações aritméticas para o cálculo das seguintes fórmulas:

   a) $c = (h + 0.5d)\ln\left(\frac{2h}{p}\right)$

   b) $z = 2e^{x \sin x\pi}$

   c) $m = 2\left(y^2 + \frac{p}{p - 1 + p^2}\right)$

   d) $s = \sqrt{\frac{\sin^{a+b} x}{a+b}}$

   e) $g = L\left[0.5\pi r^2 - r^2 \arcsin(h/r) - h(r^2 - h^2)\right]^{1/2}$

5. Considere as variáveis `A=11`, `B=5`, `C=-4` e `D=2`. Calcule as expressões abaixo.

   a) `3*modulo(A,3)-C`

   b) `2^(2*abs(C))/8`

   c) `(A/B-fix(A/B)+sign(C)+2.8)^(15/B)`

   d) `sqrt(cos(A)^2+sin(A)^2) + sin(D*%pi/4)`

   e) `(A+C)/A * round(sign(C)+D/4)-fix(D/1.5)`

6. Qual é a primeira operação a ser executada em cada um dos comandos abaixo.

a) `R + S - W`	d) `X + Y + C * D`
b) `W1 + W2 / C ^ 2`	e) `A + D + B ^ 2 + E * 3`
c) `NOTA + MEDIA/N`	f) `A * B / C * D`

7. O que é o Espaço de Trabalho?

Capítulo 2

ARQUIVOS DE SCRIPTS

Este capítulo trata de programas sequenciais em Scilab. Serão apresentados comandos de entrada e saída além de mostrar como editar e executar programas no Scilab.

2.1 COMANDO DE ENTRADA DE DADOS

Programas de computador podem solicitar dados do usuário via teclado usando o comando `input` que tem duas formas básicas (uma para números e outra para strings). A primeira delas, usada para solicitar dados numéricos, tem a seguinte forma:

```
<variavel> = input(<string>);
```

Esta função mostra o texto *<string>* e em seguida solicita que o usuário digite um número. Por fim, o número digitado é atribuído a *<variavel>*. Exemplo:

```
-->x = input("Digite um número")
Digite um número-->10
 x  =

    10.
```

A segunda forma do comando `input` é usada para solicitar dados do tipo string ao usuário. Sua forma é:

```
<variavel> = input(<string>,"s");
```

Exemplo:

```
-->a = input("Digite alguma coisa","s")

Digite alguma coisa-->Olá

 a  =

 Olá
```

2.2 COMANDOS DE SAÍDA DE DADOS

Comandos de saída de dados fornece ao usuário um meio de visualizar dados e o resultado de algum processamento. A forma mais simples de visualizar dados no Scilab é suprimir o ponto e vírgula no final do comando como mostrado na Seção 1.2.2.

```
-->x = 3;

-->y = 4;

-->r = sqrt(x*x+y*y)      // Com a omissão do ponto e virgula
 r  =                     // o resultado é exibido

    5.
```

A FUNÇÃO DISP

A função `disp()` é outra maneira de exibir dados. Por exemplo,

```
-->v0 = 2;

-->a = 4;

-->t = 3;

-->v = v0+a*t;

-->disp(v)             // disp não mostra o nome
                       // da variável
    14.
```

Este exemplo concatena dois strings e exibe o resultado:

```
-->nome = "Maria";

-->disp("Seu nome é " + nome)

 Seu nome é Maria
```

A função `disp` é freqüentemente usada em conjunto com a função `string` que converte um número em string. Por exemplo,

```
-->disp("A velocidade final é " + string(v))

 A velocidade final é 14
```

Use a função `format` para formatar a saída de dados numéricos.

A FUNÇÃO PRINTF

A função `printf` é a forma mais flexível de exibir dados porque produz uma saída formatada. Por exemplo,

`printf` é um clone do comando de mesmo nome da linguagem de programação C.

```
-->printf("Alô mundo\n");
Alô mundo
```

O caracter `\n` (chamado de *new line*) avisa ao comando `printf` para gerar uma nova linha. Mais precisamente, `\n` move o cursor para o começo da linha seguinte. Por exemplo, colocando `\n` após o string `Alô` faz com que `printf` gere uma nova linha após `Alô`:

```
-->printf("Alô\nmundo");
Alô
mundo
```

A forma geral do comando `printf` é:

```
printf(<formato>,<lista de dados>);
```

<formato> é uma string descrevendo a forma com que a lista de dados será exibida. Exemplo:

```
-->A = 2;

-->printf("A variável A contém o valor %g\n",A);

A variável A contém o valor 2
```

A símbolo %g (chamado de caractere de formatação) indica como cada variável da lista de dados será exibido dentro da string de formatação *<formato>*. Neste último exemplo, %g é substituido pelo valor da variável A no momento da impressão. No seguinte exemplo, as variáveis `A` e `B` substituirão os caracteres de formatação %g nas posições correspondentes:

```
-->A = 8/4;

-->B = A + 3;

-->printf("Os valores calculados foram %g e %g\n",A,B);
Os valores calculados foram 2 e 5
```

Mais exemplos:

```
-->printf("A = %g  B = %g",A,B);
A = 2  B = 5
-->printf("A = %g\nB = %g\n",A,B);
A = 2
B = 5

-->F = "Os valores calculados foram %g e %g";

-->printf(F,A,B);
Os valores calculados foram 2 e 5
```

Se a variavel for do tipo string, usa-se o caractere de formatação %s em vez de %g. Por exemplo:

```
-->nome = "Joao";

-->altura = 1.65;

-->printf("A altura de %s é %g",nome, altura);
A altura de Joao é 1.65
```

2.3 ARQUIVOS DE SCRIPTS

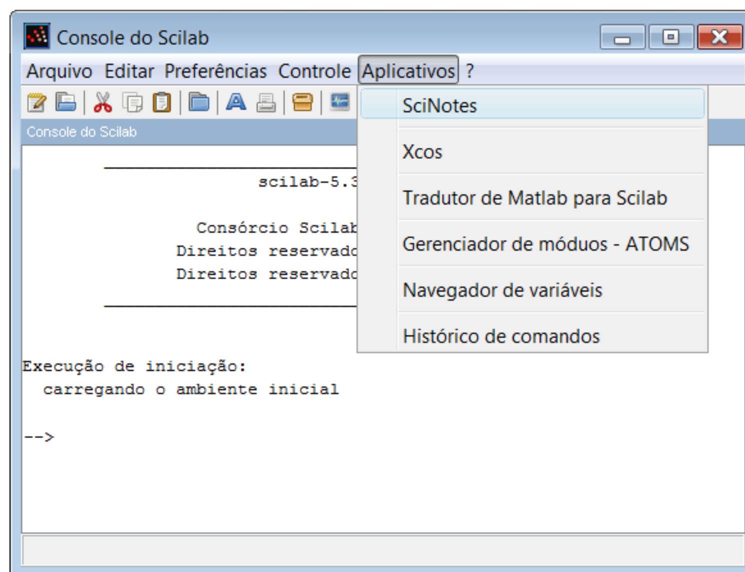
Digitamos comandos no console do Scilab para resolver problemas simples. Mas, se o número de comandos para digitar é grande, então é melhor salvar os comandos em um arquivo de script.

Um arquivo de script é um arquivo que contém um script, isto é, uma sequência de comandos para ser executada pelo computador. Os comandos do arquivo de script são executados automaticamente pelo Scilab da mesma forma que você faria se os tivesse digitado no console.

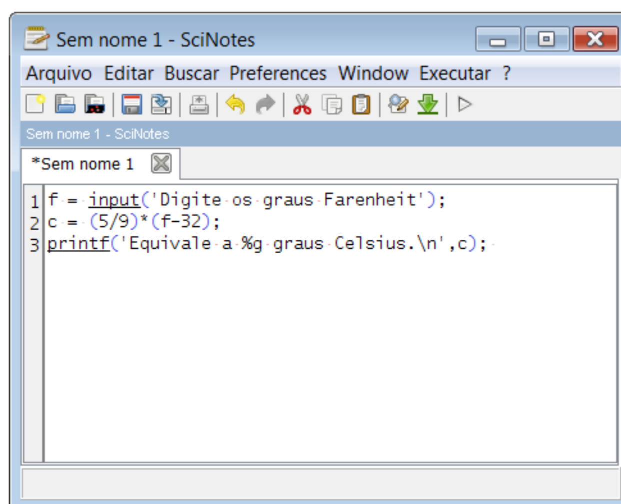
Arquivos de script são arquivos de texto puro ou plano, isto é, arquivos sem formatações como negrito, itálico e sublinhado e, por causa disso, eles podem ser criados em qualquer editor de texto. O Scilab já contém um editor de textos (chamado SciNotes) que facilita a criação de arquivos de script. Por isso é recomendável utilizar o SciNotes em vez de algum outro editor de texto. Mas se você quiser usar um editor diferente, como o Microsoft Word, tenha o cuidado de salvar os arquivos como um arquivo texto. Caso contrário, se o arquivo for salvo no formato nativo (e.g., o formato doc do Word) poderão conter caracteres especiais que causarão erros no Scilab.

2.4 CRIANDO ARQUIVOS DE SCRIPT

Para criar arquivos de scripts, selecione a opção **Aplicativos** » **SciNotes**[1] ou clique no ícone correspondente na barra de ferramentas.



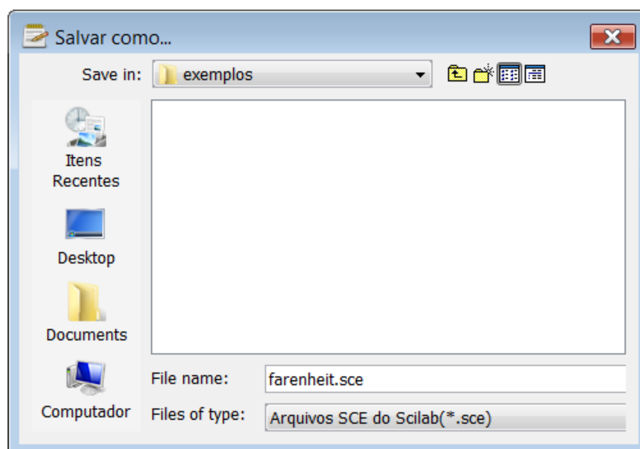
Criamos um arquivo de script digitando comandos no editor:



Note que o arquivo de script acima contém comandos para converter graus Farenheit em graus Celsius.

[1]As figuras mostradas nesta seção podem não corresponder exatamente a versão do scilab que você utilizando.

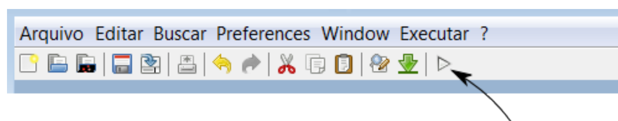
O próximo passo é salvar o arquivo de script. Selecione a opção **ARQUIVO** » **SALVAR**. Digite o nome `farenheit.sce` (os nomes de arquivos de script possuem extensão ".sce") e selecione a pasta de sua preferência para salvar o arquivo.



O usuário pode criar novos arquivos de script selecionando a opção **ARQUIVO** » **NOVO**. Os passos para executar um script são mostrados na seção seguinte.

2.5 EXECUTANDO ARQUIVOS DE SCRIPT

Para executar um script, clique no seguinte ícone da barra de ferramentas:



Alternativamente, pode-se executar um script teclando Ctrl+Shift+E, ou seja, teclando nas teclas "Ctrl", "Shift" e "E" simultaneamente ou selecionando a opção **Executar** do editor. Uma terceira forma de executar um script é a mostrado na Seção 2.8.

A Figura 2.1 mostra a execução do script `farenheit.sce` criado na Seção 2.4. O Scilab solicitou um número para fazer a conversão de graus Farenheit em graus Celsius. Digitou-se 50 e depois **ENTER**:

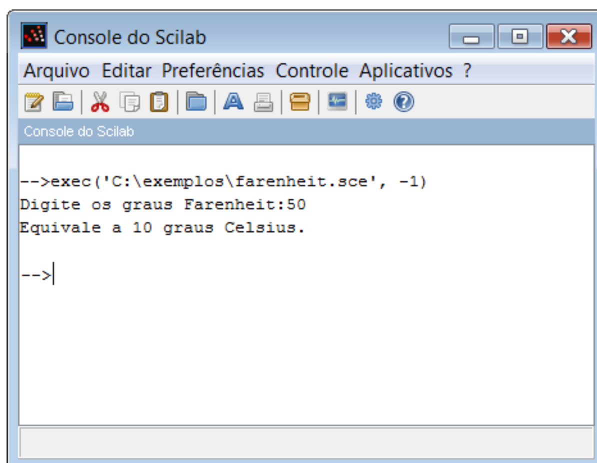


Figura 2.1: Executando um arquivo de script

Se **houver erros de digitação** no script então poderá ocorrer erros na sua execução. Neste caso, retorne ao editor e corrija o erro. Em seguida, siga os mesmos passos descritos anteriormente: salve o script (selecione **ARQUIVO** » **SALVAR**) e então execute o script (tecle Ctrl+Shift+E ou selecione **EXECUTAR**).

2.6 EXEMPLOS

Exercício resolvido 2.6.1. Escreva um programa Scilab para calcular a distância entre dois pontos (x_1, y_1) e (x_2, y_2) no plano cartesiano. Os pontos são digitados pelo usuário. A distância entre dois pontos é dada por:

$$d = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

Solução:

```
1 x1 = input("Digite X1 da primeira coordenada");
2 y1 = input("Digite Y1 da primeira coordenada");
3 x2 = input("Digite X2 da segunda coordenada");
4 y2 = input("Digite Y2 da segunda coordenada");
5 dx = x1 - x2;
6 dy = y1 - y2;
7 d = sqrt(dx*dx + dy*dy);
8 printf("A distância é %g\n",d);
```

Execute este script no Scilab. O resultado de uma possível execução seria:

Resultado

```
Digite X1 da primeira coordenada-->2
Digite Y1 da primeira coordenada-->3
Digite X2 da segunda coordenada-->5
Digite Y2 da segunda coordenada-->7
A distância é 5
```

Exercício resolvido 2.6.2. Elabore um programa Scilab para calcular a resistência equivalente entre dois registores R_1 e R_2 em paralelo. Lembre-se que a resistência equivalente entre dois registores em paralelo é dado por:

$$\frac{1}{R_{\text{eq}}} = \frac{1}{R_1} + \frac{1}{R_2}$$

Esta fórmula, também pode ser reescrita como:

$$R_{\text{eq}} = \frac{R_1 R_2}{R_1 + R_2}$$

Solução:

```
1 r1 = input("Digite o valor da primeira resistência (em ohms)");
2 r2 = input("Digite o valor da segunda resistência (em ohms)");
3 req = (r1*r2)/(r1+r2);
4 printf("A resistência equivalente é igual a %g\n",req);
```

Execute este script no Scilab.

2.7 LINHAS DE COMENTÁRIOS

Comentários podem ser inseridos em um programa para dar clareza e assim fazer as pessoas compreenderem o que nele está escrito. Comentários são inseridos após símbolo `//`. O símbolo `//` indica que o resto da linha (i.e., o comentário) deve ser ignorado pelo Scilab. Um exemplo é mostrado a seguir:

Boa programação: insira comentários para explicar o funcionamento do programa

```
1 // Programa para calcular a resistência equivalente de dois
2 // registores em paralelo.
3 r1 = input("Digite o valor da primeira resistência (em ohms)");
4 r2 = input("Digite o valor da segunda resistência (em ohms)");
5 req = (r1*r2)/(r1+r2);
6 printf("A resistência equivalente é igual a %g\n",req);
```

2.8 ALTERANDO O DIRETÓRIO DE TRABALHO

Uma forma alternativa de executar scripts é através do comando `exec`. O comando `exec` tem a forma:

```
exec(<script>)
```

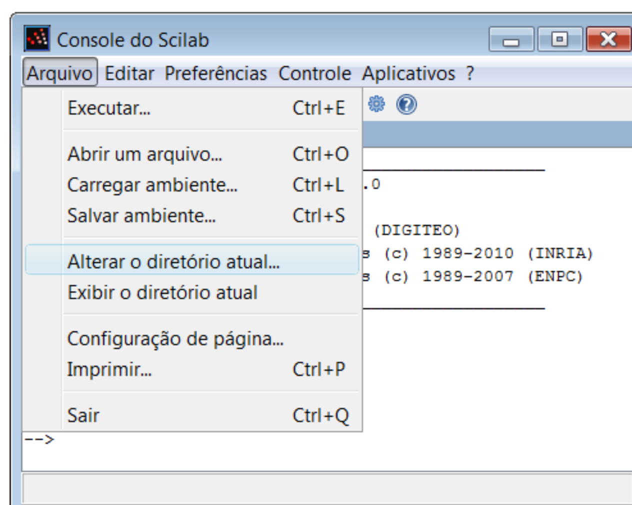
Como exemplo, considere o script `farenheit.sce` descrito na seção 2.4.

```
-->exec("farenheit.sce");
Digite os graus Farenheit-->50
Equivale a 10 graus Celsius.
```

AVISO: este comando funciona somente se o arquivo de script estiver no **diretório atual** do scilab caso contrário ocorrerá um erro como este:

```
-->exec("farenheit.sce");
                      !--error   241
O arquivo "farenheit" não existe.
```

Resolva este problema alterando o diretório atual: selecione **ARQUIVO** » **ALTERAR O DIRETÓRIO ATUAL** e depois escolha a pasta de `farenheit.sce` (i.e., `c:\exemplos`) na caixa de diálogo a seguir:



Agora o script deveria rodar sem problemas.

Alternativamente, o diretório atual pode ser alterado com o comando `chdir` (ou abreviadamente `cd`)[2]:

```
-->cd c:\exemplos;              // altera o diretório corrente

-->exec("farenheit.sce");       // executa o script
Digite os graus Farenheit-->50
Equivale a 10 graus Celsius.
```

Use o comando `pwd` para visualizar o diretório de trabalho do Scilab:

```
-->pwd
 ans  =

 c:\exemplos
```

Mesmo pertencendo a uma pasta diferente do diretório atual, um script ainda pode ser executado, desde que fornecido o caminho (do inglês, *path*) da pasta do script. Por exemplo:

```
-->cd c:\outrodir                     // alteração do diretório atual

-->exec('c:\exemplos\farenheit.sce');
Digite os graus Farenheit-->50
Equivale a 10 graus Celsius.
```

O comando `unix_w` permite a execução de qualquer comando comum do DOS (no windows) ou do *shell* (no Linux/Unix). Por exemplo, o comando DIR do DOS (utilizado para visualizar o diretório de trabalho) pode ser executado do seguinte modo:

```
-->unix_w('dir')
```

[2]Para executar um comando chdir (ou qualquer outro comando) automaticamente no inicio de uma sessão do Scilab, inclua ele no arquivo de script de configuração scilab.star.

Capítulo 3

ESTRUTURAS DE SELEÇÃO

Este capítulo introduz as expressões booleanas e as estruturas de seleção que permitem os programas tomarem decisões.

3.1 ESTRUTURAS DE CONTROLE

Os programas desenvolvidos no capítulo anterior basicamente liam (e/ou escreviam) variáveis e avaliavam expressões aritméticas atribuindo o resultado a uma variável. Os comandos de tais programas eram executados **sequencialmente** na ordem em que foram escritas, isto é, de cima para baixo e da esquerda para a direita. Vale dizer, porém, que bem poucos programas úteis podem ser construídos desta forma. De fato, é necessário adicionar dois mecanismos para tornar os programas flexíveis e poderosos:

1. As estruturas de seleção;
2. As estruturas de repetição.

A estruturas de seleção são úteis para implementar situações que requerem a execução de ações alternativas que dependem de certas condições. Exemplos:

- Se a nota do aluno for maior que 7 então avise que ele foi "aprovado"; caso contrário informe que ele está em recuperação.
- Se a lâmpada está queimada compre uma nova lâmpada; caso contrário acenda a lâmpada para ler o livro.
- Compre ou venda ações da bolsa de valores de acordo se índices econômicos sobem ou descem, respectivamente.

Portanto, a estrutura de seleção é um mecanismo para *selecionar* entre caminhos alternativos de execução comandos.

A estrutura de repetição é um mecanismo para *repetir* a execução de uma sequência de comandos. Os comandos que implementam estes mecanismos são chamados de **comandos de controle**. São exemplos de comandos de controle: `IF`, `FOR` e `WHILE` mostrados adiante. Uma **estrutura de controle** é formado por um comando de controle juntamente com o conjunto de comandos cuja execução ele controla.

Com estas três estruturas:

1. Sequência,
2. Seleção,

3. Repetição,

é possível construir qualquer programa de computador (esta é a tese principal da chamada "programação estrutura").

A estrutura de seleção será estudada neste capítulo e a estrutura de repetição no capítulo 4. O uso das estruturas de seleção requer o domínio das expressões booleanas estudadas na seção seguinte.

3.2 EXPRESSÕES BOOLEANAS

Da mesma forma que avaliar uma expressão aritmética produz um valor numérico, avaliar expressões booleanas produz um valor lógico (verdadeiro ou falso). Expressões booleanas são também chamadas de expressões lógicas.

O termo "booleano", largamente usado na computação, é uma homenagem ao matemático e lógico George Boole.

Uma expressão booleana simples usa os *operadores relacionais* para comparar expressões aritméticas:

Operadores Relacionais	
Operador	Descrição
<	Menor que
<=	Menor ou igual a
>	Maior que
>=	Maior ou igual a
==	Igual a
<> ou ~=	Diferente de

Por exemplo, suponha que a variável `A` contém o valor `5`. Então temos:

```
A > 0                 ←—Verdadeiro
A == 3 + 1            ←—Falso
2*A <> 5/10 + A       ←—Verdadeiro
```

As expressões booleanas podem ser combinadas usando os *operadores booleanos*:

Operadores Booleanos	
Operador	Descrição
&	E (conjunção)
|	Ou (disjunção não exclusiva)
~	Não (negação)

Os operadores são definidos pelas seguintes tabelas (também chamadas de *tabelas verdade*):

A	B	A & B	A | B	~A
Verdadeiro	Verdadeiro	Verdadeiro	Verdadeiro	Falso
Verdadeiro	Falso	Falso	Verdadeiro	Falso
Falso	Verdadeiro	Falso	Verdadeiro	Verdadeiro
Falso	Falso	Falso	Falso	Verdadeiro

Por exemplo, considere as seguintes atribuições `A = 5`, `B = 1`, `C = 2`, `D = 8` e `E = 3`. Então temos:

```
A > B | D > E         ←—Verdadeiro
~(A > B)              ←—Falso
A + 3 == 8 & A > B    ←—Verdadeiro
```

Erro comum de programação: programadores principiantes costumam confundir o operador "Igual a" (==) com o operador de atribuição (=).

3.3 VARIÁVEIS BOOLEANAS

Da mesma forma que conteúdo de uma variável pode ser um valor numérico ou um string, ela pode conter um valor booleano (verdadeiro ou falso). Assim,

`C = 1 > 2` ⟵C é um variável booleana com valor falso.

`A = C | 2 > 10` ⟵A é um variável booleana com valor falso.

No Scilab, duas variáveis especiais `%t` (do inglês, true) e `%f` (do inglês, false) representam o valor verdadeiro e falso, respectivamente. Exemplos:

`B = %f`
`A = %t`
`C = A | B` ⟵C contém um valor verdadeiro.

3.4 TIPOS DE DADOS PRIMITIVOS

O Scilab manipula três tipos primitivos de informação que consta na maioria das linguagens de programação tradicional:

1. O número real;
2. O tipo string;
3. O tipo booleano.

O Scilab não possue o tipo número inteiro (usado em muitas linguagens). O tipo da variável pode mudar toda vez que um valor lhe é atribuído. O novo tipo será o mesmo do último valor atribuído. Exemplo:

```
A = 2.5;              // A variável A é um número real
A = "livro";          // Agora ela é uma string
```

3.5 ORDEM DE AVALIAÇÃO ENTRE OS OPERADORES

A ordem de avaliação entre todos operadores que já foram descritos é dado na Tabela 3.1. Exemplos:

`K | P &  Q` ⟵Primeiro é realizada a operação `&`. Seguindo com a operação `|`.

`A > B | D < E` ⟵Primeiro é realizada a operação >, depois a operação <, por fim a operação |.

`~(A > B) & A < 2` ⟵Primeiro é realizada a operação >. Seguindo com a operação `~`. Depois com a operação <. Por fim com a operação `&`.

Exercício resolvido 3.5.1. Avaliar a seguinte expressão: `~(2<5)|40/5==10&6+2>5`.

Solução:

Tabela 3.1: Regras de prioridade e associatividade entre operadores

<table>
<tr><td>1^{a}</td><td>~</td><td>não</td><td>da direita para a esquerda</td></tr>
<tr><td>2^{a}</td><td>^</td><td>potenciação</td><td>da direita para a esquerda</td></tr>
<tr><td>3^{a}</td><td>*
/</td><td>multiplicação
divisão</td><td>da esquerda para a direita</td></tr>
<tr><td>4^{a}</td><td>+
-</td><td>Adição
Subtração</td><td>da esquerda para a direita</td></tr>
<tr><td>5^{a}</td><td><
<=
>
>=</td><td>Menor que
Menor ou igual a
Maior que
Maior ou igual a</td><td>da esquerda para a direita</td></tr>
<tr><td>6^{a}</td><td>==
<> ou ~=</td><td>Igual a
Diferente de</td><td>da esquerda para a direita</td></tr>
<tr><td>7^{a}</td><td>&</td><td>e</td><td>da esquerda para a direita</td></tr>
<tr><td>8^{a}</td><td>|</td><td>ou</td><td>da esquerda para a direita</td></tr>
</table>

```
~ ( 2 < 5 ) | 40 / 5 == 10 & 6 + 2 > 5
   ~ %t     | 40 / 5 == 10 & 6 + 2 > 5
     %f     | 40 / 5 == 10 & 6 + 2 > 5
     %f     |   8    == 10 & 6 + 2 > 5
     %f     |   8    == 10 & 8 > 5
     %f     |   8    == 10 & %t
     %f     |  %f          & %t
     %f     |        %f
           &f
```

3.6 A SELEÇÃO SIMPLES IF-END

Caracteriza-se por permitir a execução de uma sequência de comandos quando certas condições, representadas por expressões booleanas, forem satisfeitas. A seleção simples tem a seguinte forma:

```
if <expressão booleana>
    <sequência de comandos>
end
```

A sequência de comandos só será executada se a expressão booleana retornar um valor verdadeiro.

Exercício resolvido 3.6.1. Elaborar um programa para escrever a média de duas notas. Se a média for maior que sete, o programa deverá também escrever Parabéns.

Solução:

```
1  nota1 = input("digite a primeira nota");
2  nota2 = input("digite a segunda nota");
3  media = (nota1+nota2)/2;
4  printf("Sua média é %g\n",%g)
5  if media > 7
6    printf("Parabéns!");
7  end
```

Resultado

```
digite a primeira nota-->7.5
digite a segunda nota-->8.1
Sua média é 7.8
Parabéns!
```

Comentário. Este programa escreve a média do aluno, mas só executa a linha 6 se sua nota for maior que 7.

3.7 A SELEÇÃO BIDIRECIONAL IF-ELSE-END

Caracteriza-se por selecionar entre duas sequências de comandos quando certas condições, representadas por expressões booleanas, forem satisfeitas. A seleção bidirecional tem a seguinte forma:

```
if <expressão booleana>
    <primeira sequência de comandos>
else
    <segunda sequência de comandos>
end
```

A primeira sequência de comandos será executada se a expressão booleana devolver um valor verdadeiro, caso contrário a segunda sequência de comandos será executada. Os seguintes exemplos ilustram o comando de seleção bidirecional.

Boa programação: Sempre use *endentação*, ou seja, acrescente dois espaços antes dos comandos que estão aninhados no comando IF para melhorar a legibilidade do programa.

Exercício resolvido 3.7.1. Elaborar um programa para ler quatro notas, calcular a média e informar se o aluno passou de ano (aprovado) ou não (reprovado). A média para passar de ano é 6.

Solução:

```
1  // Leitura das notas
2  nota1 = input("Digite a 1a. nota bimestral");
3  nota2 = input("Digite a 2a. nota bimestral");
4  nota3 = input("Digite a 3a. nota bimestral");
5  nota4 = input("Digite a 4a. nota bimestral");
6  media = (nota1 + nota2 + nota3 + nota4)/4;
7  printf("Media = %g\n",media);  // Calculo da media anual
8  if media > 6
9    printf("Aluno aprovado\n");
10 else
11   printf("Aluno reprovado\n");
12 end
```

Resultado

```
Digite a 1a. nota bimestral-->8.4
Digite a 2a. nota bimestral-->7.3
Digite a 3a. nota bimestral-->9.1
Digite a 4a. nota bimestral-->8.5
Media = 8.325
Aluno aprovado
```

Comentário. O comando IF testa se a média é maior que 6,0. Se sim, o programa executa o comando da linha 9 que escreve Aluno aprovado. Caso contrário, o programa executa a linha 11 que escreve Aluno reprovado.

Exercício resolvido 3.7.2. Elaborar um programa para calcular o valor da função

$$f(x) = \begin{cases} x^2 + 16 & \text{se } x \geq 0 \\ 0 & \text{se } x < 0 \end{cases}$$

dado um valor x fornecido via teclado.

Solução:

```
1  x = input("Entre com o valor de x");
2  if x >= 0
3    f = x^2+16;
4  else
5    f = 0;
6  end;
7  printf("O valor da função é %g\n",f);
```

Resultado

```
Entre com o valor de x-->2
O valor da função é 20
```

3.8 ANINHANDO SELETORES

Comandos de seleção podem ser aninhados de diversas formas criando ampla variedade de construções como mostra os exemplos a seguir.

Exercício resolvido 3.8.1. Elaborar um programa para escrever a situação do aluno. O aluno com média maior ou igual a 7,0 será aprovado. O aluno com média entre 5,0 e 7,0 ficará em recuperação. Com média abaixo de 5,0, o aluno será reprovado.

Solução:

```
1  // Leitura das notas
2  nota1 = input("Digite a 1a. nota bimestral");
3  nota2 = input("Digite a 2a. nota bimestral");
4  nota3 = input("Digite a 3a. nota bimestral");
5  nota4 = input("Digite a 4a. nota bimestral");
6  media = (nota1 + nota2 + nota3 + nota4)/4;
7
8  printf("Media = %g\n",media);  // Calculo da media anual
9
10 if media >= 7
11   printf("Aprovado\n");
12 else
13   if media >= 5
14     printf("Recuperação\n");
15   else
16     printf("Reprovado\n");
17   end
18 end
```

Resultado

```
Digite a 1a. nota bimestral-->7.3
Digite a 2a. nota bimestral-->6.5
Digite a 3a. nota bimestral-->5.5
Digite a 4a. nota bimestral-->7.5
Media = 6.7
Recuperação
```

Comentário. Se a média for maior ou igual que 7 o programa escreve Aprovado (linha 11). Caso contrário o programa executa o segundo `IF` (após o `else`). Aqui, temos um novo teste `IF` (aninhado dentro de primeiro `IF`) que fará o programa escrever Recuperação se média for maior ou igual a 5 (linha 14) ou, caso contrário, Reprovado (linha 16).

Exercício resolvido 3.8.2. Escreva um programa que leia três números e escreva o menor deles.

Solução:

```
1  // Leitura dos números
2  a = input("Digite um numero");
3  b = input("Digite um numero");
4  c = input("Digite um numero");
5
6  // Determina o menor número
7  if a < b & a < c
8    menor = a;
9  else
10   if b < c
11     menor = b;
12   else
13     menor = c;
14   end
15 end
16 printf("O menor número é %g\n",menor);
```

Resultado

```
Digite um numero-->2
Digite um numero-->3
Digite um numero-->1
O menor número é 1
```

Vejamos dois casos um pouco mais complicados.

Exercício resolvido 3.8.3. Elaborar um programa Scilab para ler três medidas a, b e c. Depois verificar se elas podem ser as medidas dos lados de um triângulo. Se forem, verificar ser se o triângulo é equilátero, isósceles ou escaleno.

Solução:

Para saber se três medidas podem ser as medidas dos lados de um triângulo usamos o seguinte propriedade da Geometria (conhecido como desigualdade triangular):

"Em todo triângulo, cada lado é menor que a soma dos outros dois".

Portanto, se as medidas satisfazem a desigualdade triangular acima então elas formam um triângulo, caso contrário não formam um triângulo.

Relembrando, também, que os triângulos se classificam em:

1. Equilátero se tem três lados de medidas iguais.
2. Isósceles se tem dois lados de medidas iguais.
3. Escaleno se dois quaisquer lados não possuem medidas iguais.

Com as informações acima, elaboramos o seguinte programa:

```
1  // Leitura das medidas
2  a = input("Digite a primeira medida");
3  b = input("Digite a segunda medida");
4  c = input("Digite a terceira medida");
5
6  if a < b + c & b < a + c & c < a + b // Verifica se as medidas
7                                       // formam um triângulo.
8    if a == b & b == c
9      printf("Triângulo equilátero");  // Três lados iguais.
10   else
11     if a == b | a == c | b == c
12       printf("Triângulo isósceles"); // Dois lados iguais.
13     else
14       printf("Triângulo escaleno");  // Todos os lados são diferentes.
15     end
16   end
17
18 else
19   printf("Não é um triângulo\n");    // Não satisfez a propriedade da
20                                      // desigualdade triangular.
21 end
```

Resultado

```
Digite a primeira medida-->5
Digite a segunda medida-->5
Digite a terceira medida-->3
Triângulo isósceles
```

Resultado

```
Digite a primeira medida-->5
Digite a segunda medida-->3
Digite a terceira medida-->1
Não é um triângulo
```

Comentário. Após a leitura das medidas a, b e c (linhas 2-4), o programa verifica, na linha 6, se estas medidas satisfazem a desigualdade triangular. Se sim, o programa classifica o triângulo formado por a, b, e c (linhas 8-16) do seguinte modo. Na linha 8, se a expressão booleana "`a == b & b == c`" retornar verdadeiro então todos os lados são iguais e portanto o triângulo é equilátero. Caso contrário, o programa segue adiante para classificar o triângulo em isósceles ou escaleno. Se, porém, as medidas não satisfazem a desigualdade triangular da linha 6, então o programa não executa as linhas 8-16, e simplesmente escreve que a medidas não formam um triângulo (linha 19).

Exercício resolvido 3.8.4. (**Equação do segundo grau**). Dados os três coeficientes a, b, c de uma equação do segundo grau $ax^2 + bx + c = 0$, elaborar um algoritmo para calcular suas raízes. O discriminante da equação é dado por $\Delta = b^2 - 4ac$. Para cada situação da seguinte tabela, o algoritmo tomará a respectiva ação:

Situação	Ação
$a = 0$ e $b = 0$	Escrever que a equação é degenerada.
$a = 0$ e $b \neq 0$	Calcular e escrever a única raiz $x = -c/b$.
$a \neq 0$ e $c = 0$	Calcular e escrever as duas raízes: $x_1 = 0$, $x_2 = -b/a$
$a \neq 0$ e $c \neq 0$ e $\Delta \geq 0$	Calcular e escrever as duas raízes: $x_1 = \frac{-b+\sqrt{\Delta}}{2a}$, $x_2 = \frac{-b-\sqrt{\Delta}}{2a}$
$a \neq 0$ e $c \neq 0$ e $\Delta < 0$	As raízes são complexas. Escrever as partes real e imaginária das duas raízes.

Solução:

```
1  //
2  // Programa para calcular as raízes de uma equação do 2o grau
3  //
4
5  a=input("Digite o coeficiente a :");
6  b=input("Digite o coeficiente b :");
7  c=input("Digite o coeficiente c :");
8
9  if (a==0) & (b==0)       // Equacao degenerada.
10   printf("Equacao degenerada\n");
11 end
12
13 if (a==0) & (b<>0)       // Equação do 1o grau
14   printf("Raiz única em %g.\n",-c/b);
15 end
16
17 if (a<>0) & (c==0)       // Equacao do 2o grau com raizes reais em 0 e -b/a
18   x = -b\a;
19   printf("Raiz1 = 0\n");
20   printf("Raiz2 = %g\n",x);
21 end
22
23 if (a<>0) & (c<>0)       // Equacao do 2o grau
24
25   disc = b*b - 4*a*c;    // Cálculo do discriminante
26
27   if disc >= 0           // Teste do discriminante
28     // Raizes reais.
29     x1 = -b/(2*a) + sqrt(disc)/(2*a);
30     x2 = -b/(2*a) - sqrt(disc)/(2*a);
31     printf("Raiz1 = %g\n",x1);
32     printf("Raiz2 = %g\n",x2);
```

```
33     else
34       // Raizes complexas
35       pr = -b/(2*a);
36       pi = sqrt(abs(disc))/(2*a);
37       printf("Parte Real = %g\n",pr);
38       printf("Parte Imaginária = %g\n",pi);
39     end
40   end
```

Resultado
```
Digite o coeficiente a :-->1
Digite o coeficiente b :-->-5
Digite o coeficiente c :-->6
Raiz1 = 3
Raiz2 = 2
```

Resultado
```
Digite o coeficiente a :-->0
Digite o coeficiente b :-->5
Digite o coeficiente c :-->10
Raiz única em -2.
```

Resultado
```
Digite o coeficiente a :-->2
Digite o coeficiente b :-->3
Digite o coeficiente c :-->6
Parte Real = -0.75
Parte Imaginária = 1.56125
```

O Scilab manipula números complexos automaticamente e por isso não era necessário se preocupar em manipular os números complexos como foi feito no exemplo anterior. O mesmo não acontece com algumas linguagens tradicionais. Por exemplo, em linguagem C, devemos tomar cuidado com números complexos pois se o discriminante da equação for um número negativo então a raiz quadrada gera um erro. Por exemplo, em C uma declaração equivalente a esta

```
x1 = -b/(2*a) + sqrt(disc)/(2*a);
x2 = -b/(2*a) - sqrt(disc)/(2*a);
```

produzia um erro se o valor da variável `disc` fosse negativo. O Scilab não gera erro ao calcular a raiz quadrada de um número negativo. Ao invés disso, o Scilab produz um número complexo automaticamente. A seguir é mostrado uma outra versão do programa do exemplo anterior (desta vez sem se preocupar se o discriminante é negativo ou não):

```
1    //
2    // Programa para calcular as raízes de uma equação do 2o grau
3    // Esta versão manipula os números complexos diretamento.
4
5    a=input("Digite o coeficiente a :");
6    b=input("Digite o coeficiente b :");
7    c=input("Digite o coeficiente c :");
8
9    if (a==0) & (b==0)       // Equacao degenerada.
10     printf("Equacao degenerada\n");
11   end
12
13   if (a==0) & (b<>0)       // Equação do 1o grau
14     printf("Raiz única em %g.\n",-c/b);
15   end
```

```
16
17 if (a<>0) & (c==0)       // Equacao do 2o grau com raizes reais em 0 e -b/a
18   x = -b\a;
19   printf("Raiz1 = 0\n");
20   printf("Raiz2 = %g\n",x);
21 end
22
23 if (a<>0) & (c<>0)       // Equacao do 2o grau
24   disc = b*b - 4*a*c;    // Cálculo do discriminante
25   x1 = -b/(2*a) + sqrt(disc)/(2*a);
26   x2 = -b/(2*a) - sqrt(disc)/(2*a);
27   printf("Raiz1 =");
28   disp(x1);
29   printf("Raiz2 =");
30   disp(x2);
31   if isreal(x1)
32     printf("As raízes são reais");
33   else
34     printf("As raízes são complexas");
35   end
36 end
```

Comentário. O comando de saída de dados `disp` (linhas 28 e 30) foi usado porque o comando `printf` não imprime números complexos. Para saber se as raízes são reais ou complexas, foi utilizado uma função especial `isreal()` do Scilab que devolve verdadeiro (`%t`) se seu argumento for um número real, ou falso (`%f`) se seu argumento for um número complexo. Exemplos:

```
-->isreal(1+3*%i)
 ans  =

  F

-->isreal(3)
 ans  =

  T
```

Portanto, as linhas 31-34 deste exemplo:

```
  if isreal(x1)
    printf("As raízes são reais");
  else
    printf("As raízes são complexas");
  end
```

escreveram "As raízes são reais" se `x1` for um número real, caso contrário, escreveram "As raízes são complexas".

Capítulo 4

ESTRUTURAS DE REPETIÇÃO

4.1 LAÇOS

Computadores são frequentemente usados para repetir uma mesma operação muitas vezes. Para fazer isso, utiliza-se uma estrutura de repetição. Ela faz com que um conjunto de comandos seja executado zero, uma ou mais vezes. A estrutura de repetição é, também, chamado de **laço** (do inglês, *loop*). O conjunto de comandos que se repete em um laço é denominado de **corpo do laço**. Há dois tipos de laço no Scilab:

1. Laço controlado logicamente;
2. Laço controlado por contador.

No laço controlado logicamente, os comandos (i.e., o seu corpo) são repetidos indefinidamente *enquanto* uma certa expressão booleana for satisfeita. No laço controlado por contador, os comandos são repetidos um número *predeterminado* de vezes.

Denomina-se **iteração** a repetição de um conjunto de comandos. Portanto, cada execução do corpo do laço, juntamente com a condição de terminação do laço, é uma iteração.

Nas seções seguintes serão estudados as estruturas de repetição do Scilab.

4.2 LAÇO CONTROLADO LOGICAMENTE

O laço `while` é um laço controlado logicamente. Ele repete a execução de um conjunto de comandos (o seu corpo), mas verificando antes de executar os comandos se é permitido repeti-los ou não. O laço `while` tem a seguinte forma:

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

Enquanto a <expressão booleana> for verdadeira o <conjunto de comandos> é repetido indefinidamente. No momento em que <expressão booleana> for falsa o <conjunto de comandos> não será mais repetido. Vale salientar que se <expressão booleana> for falsa da primeira vez, o <conjunto de comandos> jamais será executado.

A elaboração de programas com laços envolve, frequentemente, o uso de duas variáveis com funções especiais: os contadores e os acumuladores na qual serão mostradas a seguir. Considere os seguintes cálculos no console do Scilab:

```
-->i = 1
 i  =
```

```
    1.

-->i = i + 1
 i  =

    2.

-->i = i + 1
 i  =

    3.

-->i = i + 1
 i  =

    4.
```

Note que cada vez que a expressão `i = i + 1` é executada o valor da variável `i` é incrementado de um. A variável `i` da expressão `i = i + 1` é chamada de **contador**. Os exemplos a seguir ilustram o uso do contador em laços `while`.

Exercício resolvido 4.2.1. Calcular a média das notas de cinco alunos.

Solução: este programa realiza a tarefa de ler as notas de um aluno e calcular a sua média. O programa deverá repetir esta tarefa cinco vezes (usando `while`). O contador é usado para contar o número de repetições.

```
1   i = 1;
2   while i <= 5
3     nota1 = input("Digite a 1a. nota bimestral");
4     nota2 = input("Digite a 2a. nota bimestral");
5     nota3 = input("Digite a 3a. nota bimestral");
6     nota4 = input("Digite a 4a. nota bimestral");
7     media = (nota1 + nota2 + nota3 + nota4)/4;
8     printf("Media = %g\n",media);
9     i = i + 1;
10  end
11  printf("Fim do programa");
```

Resultado

```
Digite a 1a. nota bimestral-->7.5
Digite a 2a. nota bimestral-->8.2
Digite a 3a. nota bimestral-->8.3
Digite a 4a. nota bimestral-->7.2
Media = 7.8
Digite a 1a. nota bimestral-->6.5
Digite a 2a. nota bimestral-->6.2
Digite a 3a. nota bimestral-->8.2
Digite a 4a. nota bimestral-->7.0
Media = 6.975
Digite a 1a. nota bimestral-->8.2
Digite a 2a. nota bimestral-->7.8
Digite a 3a. nota bimestral-->4.8
Digite a 4a. nota bimestral-->8.3
Media = 7.275
Digite a 1a. nota bimestral-->6.5
Digite a 2a. nota bimestral-->7.1
Digite a 3a. nota bimestral-->8.3
```

```
Digite a 4a. nota bimestral-->6.8
Media = 7.175
Digite a 1a. nota bimestral-->9.1
Digite a 2a. nota bimestral-->8.5
Digite a 3a. nota bimestral-->9.3
Digite a 4a. nota bimestral-->9.5
Media = 9.1
Fim do programa
```

Comentário.

O programa começa inicializando a variável `i` com o valor um (linha 1). Por causa disso, a expressão `i <= 5` do laço `while` é verdadeira. Então o corpo do laço é executado pela primeira vez (primeira iteração). O laço `while` incrementa o valor da variável `i` com o valor de um toda vez que o corpo do laço (linhas 3-9) é executado. Depois que o corpo do laço tem sido executado cinco vezes (ou seja, após cinco iterações) a variável `i` possui valor seis e a expressão `i <= 5` é falsa. Por causa disso o laço termina e o programa passa a executar a linha seguinte imediatamente após o fim do laço (que é a linha 11).

Agora, considere os seguintes cálculos executados no console do Scilab:

```
-->soma = 20
 soma  =

    20.

-->x = 2.5
 x  =

    2.5

-->soma = 0
 soma  =

    0.

-->soma = soma + x
 soma  =

    2.5

-->soma = soma + x
 soma  =

    5.
```

Qual o valor da variável `soma` após repetir a execução do último comando mais duas vezes?

A variável `soma` da expressão `soma = soma + x` é chamada de **acumulador**. Vejamos um exemplo.

Exercício resolvido 4.2.2. Elaborar um programa para calcular a soma de todos os inteiros entre 1 e 100.

Solução:

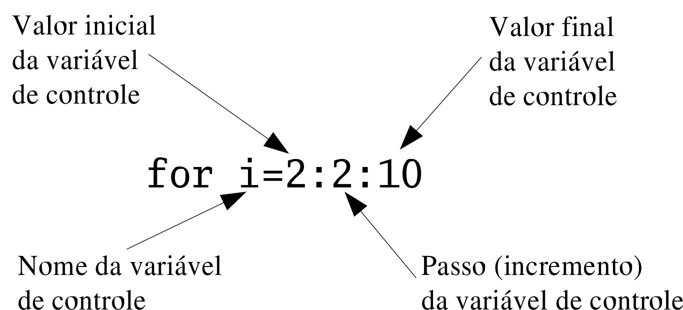


Figura 4.1: O Laço `FOR`

```
1  cont= 1;
2  soma = 0;
3  while cont <= 100
4    soma = soma + cont;
5    cont = cont + 1;
6  end
7  printf("Soma total = %g\n",soma);
```

Resultado

```
Soma total = 5050
```

Comentário. O programa inicializa as variáveis `cont` e `soma` com 1 e 0, respectivamente (porque?). Cada vez que o corpo do laço é executado, o valor de `soma` é incrementado com o valor corrente da variável `cont`. Deste modo, a variável `soma` assume os seguintes valores a cada repetição do laço.

Boa programação: sempre inicialize os contadores e acumuladores.

	Valor da variável `soma`
primeira iteração	1
segunda iteração	3
terceira iteração	6
⋮	⋮
última iteração	5050

4.3 LAÇO CONTROLADO POR CONTADOR

O comando `FOR` é o laço controlado por contador do Scilab. Ele repete a execução de um conjunto de comandos (i.e., o seu corpo) um número pré-determinado de vezes. Na forma básica, o laço `FOR` possui o nome de uma variável (chamada de **variável de controle**) e especifica seu valor inicial e final e, opcionalmente, o tamanho do passo (ou incremento) da variável de controle. Ver a Figura 4.1.

O seguinte programa escreve a palavra disciplina 10 vezes:

```
for i=1:10
  printf("disciplina\n");
end
```

Aqui a variável de controle i assume, inicialmente, o valor um e o `printf("disciplina\n")` é executado. Note que o tamanho do passo foi omitido neste exemplo. Quando isto ocorre *o passo é igual a um*. Um passo igual a um indica que a variável de controle é incrementada de um em cada iteração. O laço `for`, repete a execução de `printf("disciplina\n")` e a variável de controle i é incrementada para o valor 2. Da terceira vez que `printf("disciplina\n")`

é executado, a variável de controle i é incrementada para o valor 3 e assim por diante até alcançar o valor 10 que é o valor final. Outros exemplos seguem.

Exercício resolvido 4.3.1. Elabore um programa para escrever todos os números pares inteiros entre 1 e 50.

Solução:

```
for i=2:2:50
  printf("%g ",i);
end
```

Resultado

```
2 4 6 8 10 12 14 16 18 20 22 24 26 28 30 32 34 36 38 40 42 44 46 48 50
```

Comentário. O primeiro número par é 2, logo o variável de controle deve iniciar com 2. Porque os números devem ser escritos de dois em dois, o passo deve ser igual a 2. O valor final na variável de controle é, obviamente, 50.

Exercício resolvido 4.3.2. Elabore um programa para calcular 5! (fatorial de 5)

```
fat = 1;
for cont=2:5
  fat = fat * cont;
end
printf("O fatorial de 5 é igual a %g\n",fat);

```

Resultado

```
O fatorial de 5 é igual a 120
```

Comentário. Note que a cada interação do laço, a variável `fat` assume os seguintes valores

	Valor da variável `fat`
início	1
primeira iteração	2
segunda iteração	6
terceira iteração	24
quarta iteração	120

4.4 EXEMPLOS COM LAÇOS

Exercício resolvido 4.4.1. **Algoritmo para calcular o fatorial**.

Elabore um programa para calcular o fatorial para qualquer valor n fornecido pelo usuário. Sabendo que:

- $N! = 1 \times 2 \times 3 \times \ldots \times (N-1) \times N$;
- $0! = 1$, por definição.

Solução:

```
n = input("Entre com um número");
fat = 1;
for cont=2:n
  fat = fat * cont;
end
printf("O fatorial de %g é igual a %g\n",n,fat);
```

Resultado

```
Entre com um número-->8

O fatorial de 8 é igual a 40320
```

Exercício resolvido 4.4.2. **Exemplo de Somatório**.

Elabore um programa que calcule e escreve o valor de S.

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

Solução: note que há uma relação entre numerador e denominador da seguinte forma:

$$2 \times \text{denominador} - 1 = \text{numerador}$$

Usaremos esta relação para calcular cada termo da série no seguinte programa:

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

Resultado

```
Valor de S = 95.5008
```

Exercício resolvido 4.4.3. **O Algoritmo de Euclides**.

O Algoritmo de Euclides está presente no quinto livro de Euclides escrito por volta de 300 anos antes de Cristo. Sua finalidade é calcular o Máximo Divisor Comum (M.D.C.). Conhecido por muitos estudantes, o Algoritmo de Euclides desempenha importante papel na matemática, e por isso é interessante estudá-lo. Para calcular o M.D.C. entre dois números segue-se o algoritmo:

Passo 1. Leia duas variáveis a e b

Passo 2. r = o resto da divisão entre a e b

Passo 3. $a = b$

Passo 4. $b = r$

Passo 5. Se o valor de r é zero então a é o M.D.C. procurado e o programa termina; caso contrário volte ao passo 2.

Seguindo este algoritmo manualmente, com um lápis e papel, é possível calcular o M.D.C. entre 544 e 119, escrevendo a seguinte tabela:

a	b	$r(resto)$
544	119	68
119	68	51
68	51	17
51	17	0
17	0	

O seguinte programa Scilab implementa o algoritmo de Euclides:

```
1  a = input("Digite um número");
2  b = input("Digite um número");
3  r = 1;
4  while r <> 0
5    r = modulo(a, b);
6    a = b;
7    b = r;
8  end
9  printf("O M.D.C. é %g\n",a);
```

Exercício resolvido 4.4.4. **Usando um valor sentinela para interromper um laço**

Elabore um programa que leia via teclado um conjunto de valores inteiros e positivos. O final do conjunto é conhecido pelo valor -1. Determine o maior valor deste conjunto.

```
1  n = input("Digite um número inteiro positivo");
2  valormax = n;
3  while n <> -1
4    if n > valormax
5      valormax = n;
6    end
7    n = input("Digite um número inteiro positivo");
8  end
9  printf("Valor máximo = %g\n",valormax);
```

Comentário. A cada iteração a variável `valormax` armazena o maior valor dos números digitados até então. Portanto, no final, `valormax` armazenará o maior valor do conjunto.

Resultado

```
Digite um número inteiro positivo-->28
Digite um número inteiro positivo-->15
Digite um número inteiro positivo-->81
Digite um número inteiro positivo-->34
Digite um número inteiro positivo-->3
Digite um número inteiro positivo-->-1
Valor máximo = 81
```

Exercício resolvido 4.4.5. **A série de Fibonacci.**

A série de Fibonacci se define como tendo os dois primeiros elementos iguais a um e cada elemento seguinte é igual a soma dois elementos imediatamente anteriores. Exemplo, $1, 1, 2, 3, 5, 8\ldots$

Pede-se que escreva todos os elementos da série de Fibonacci menor ou igual a N. O valor de N é fornecido pelo teclado.

Solução:

```
1  n = input("Digite um numero inteiro positivo");
2  printf("Numeros de fibonacci menor ou igual a %g\n",n);
3  a = 1;
4  b = 1;
5  printf("%g %g ",a,b);   // imprime os dois primeiros elementos
6  c = a + b;              // calcula o proximo elemento
7  while (c <= n)
8    printf("%g ",c);
9    a = b;
10   b = c;
11   c = a + b;            // calcula o proximo elemento
12 end
```

Resultado

```
Digite um numero inteiro positivo-->1000
Numeros de fibonacci menor ou igual a 1000
1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987
```

4.5 LAÇOS ANINHADOS

Considere o seguinte programa:

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

Este programa escreve quatro vezes o caracter `x` em uma linha como segue:

```
xxxx
```

Pode-se usar laços aninhados para produzir diversas linhas de `xxxx`, conforme ilustra do no seguinte exemplo:

Exercício resolvido 4.5.1. O seguinte programa gera uma sequencia de quatro `x`'s por linha. O número de linhas é digitado pelo usuário.

```
1  lin = input("Quantas linhas");
2  for i = 1:lin
3    for j = 1:4
4      printf("x");
5    end
6    printf("\n");   // mudança da linha
7  end
```

Resultado

```
Quantas linhas-->5
xxxx
xxxx
xxxx
xxxx
xxxx
```

Comentário. Este programa solicita ao usuário o número de linhas. Neste exemplo, o usuário digitou 5 linhas. O programa, então, escreve 5 linhas e em cada linha ele repete 4 vezes o caractere `x` do seguinte modo. Inicialmente, o variável `i` assume o valor 1 na linha 2. Em seguida, a variável `j` assume o valor 1. O laço interno das linhas 3-5 é repetido 4 vezes escrevendo 4 `x´s`, ou seja, `xxxx`. Quando o laço interno termina (após

4 repetições), o comando `printf("\n")` da linha 6 cria uma nova linha. Em seguida, o `end` da linha 7 é encontrada e o programa retorna para a linha 2. Na linha 2, o valor da variável `i` é atualizado para 2. Em seguida, o programa executa novamente o laço interno e o programa escreve novamente `xxxx`. Novamente, o programa executa o comando `printf("\n")` gerando uma nova linha e o programa retorna a linha 2, onde a variável `i` é atualizado para 3. Este laço continua até a variável `i` ser igual a 5.

O próximo programa escreve a tabuada de 1 até 10. Este programa usa o string de formatação `%f` em vez de `%g` para que os dados permaneçam alinhados na tabela da tabuada. O string de formatação `%f` permite especificar o número de caracteres que será ocupado pelo dado escrito com `printf` e também o número de dígitos depois do ponto decimal. Por exemplo, o string `%5.2f` escreva um número ocupando cinco caracteres com dois dígitos depois do ponto decimal. Exemplo:

```
-->a = 223.345547867783;

-->printf("%5.2f",a);
23.35
```

O seguinte exemplo, escreve a variável `a` ocupando 10 caracteres com dois dígitos após o ponto decimal:

```
-->printf("%10.2f",a);
     23.35
```

Usando cinco dígitos após o ponto decimal:

```
-->printf("%10.5f",a);
  23.34555
```

Exercício resolvido 4.5.2. Elaborar um programa para escrever a tabuada de 1 até 10.

```
1  printf("     1    2    3    4    5    6    7    8    9   10\n");
2  printf("   ------------------------------------------------\n");
3  for i=1:10
4    printf("%2.0f ",i);
5    for j = 1:10
6      printf("%3.0f  ",i*j);
7    end
8    printf("\n");
9  end
```

Resultado

```
     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
```

Comentário. Inicialmente, a variável `i` assume o valor um na linha 3. Na linha 4, o programa escreve o número um para indicar que a tabuada de um será escrita nesta

mesma linha. Em seguida, a variável `j` assume o valor um. O laço interno das linhas 5-7 é repetido 10 vezes escrevendo a tabuada de um. Quando o laço interno termina (após 10 repetições), o comando `printf("\n")` da linha 8 gera uma nova linha. O programa retorna para a linha 3. Na linha 3, o valor da variável `i` é atualizado para dois. Em seguida, o programa executa novamente o laço interno e o programa escreve a tabuada de dois. Novamente, o programa executa o comando `printf("\n")` gerando uma nova linha e o programa retorna a linha 3, onde a variável `i` é atualizado para três. A tabuada de três é então escrita. Laço externo continua até a variável `i` ser igual a dez.

Capítulo 5

MATRIZES

Matrizes são agregados de dados dentro de uma mesma variável. Matrizes são agregados de dados homogêneos no sentido de que os dados têm sempre *mesmo tipo de conteúdo*, ou seja, uma matriz pode conter somente dados numéricos ou somente strings, mas não os dois simultaneamente [1]. No jargão da informática, agregados de dados homogêneos são frequentemente chamados de *arrays* ou arranjos. Preferimos o termo matriz (ao invés de *array*) porque é mais usual no jargão da matemática.

5.1 VETORES

Matrizes unidimensionais são chamados de **vetores**. Em um vetor é possível armazenar vários itens em uma única variável. Na Figura 5.1 é mostrada uma variável `nota` contendo as notas de alunos. Os itens contidos em um vetor são chamados de **elementos** do vetor. Portanto, o vetor `nota` possui dez elementos. Seus elementos podem ser acessados individualmente. Por exemplo, `nota(4)` refere-se ao quarto elemento do vetor nota. O valor entre os parênteses de `nota(4)` é chamado de **índice** ou **subscrito** e é usado para individualizar um elemento do vetor.

Vetores podem ser construídos usando os colchetes [e]. Os elementos são envolvidos por colchetes e separados por espaços (ou vírgula). Exemplo,

```
-->nota = [8.1 5.2 9.2 7.2 6.5 5.2 8.5 9.5 6.5 10.0];
```

Os elementos do vetor são acessados da seguinte forma:

```
-->nota(2)
 ans  =

    5.2

-->nota(5)
 ans  =

    6.5

-->nota(8)
 ans  =

    9.5
```

Pode-se somar as três primeiras notas do seguinte modo:

[1] O Scilab também dispõe de agregados de dados heterogêneos que são chamados de listas e podem armazenar simultaneamente dados numéricos e strings. Listas não serão estudados neste capítulo.

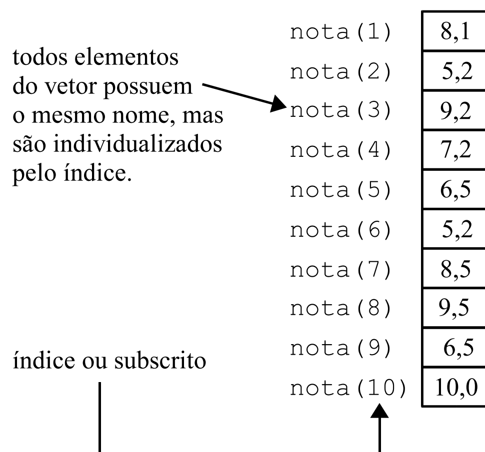


Figura 5.1: Vetor com dez elementos

```
-->nota(1) + nota(2) + nota(3)
 ans  =

    22.5
```

Uma aplicação de vetores é armazenar variáveis indexadas da matemática. Por exemplo, as variáveis indexadas $x_1 = 2$, $x_2 = 4$, $x_3 = 2$ e $x_4 = 3$ são armazenadas como:

```
-->x(1) = 2;

-->x(2) = 4;

-->x(3) = 2;

-->x(4) = 3;
```

ou equivalentemente como

```
-->x = [2 4 2 3];
```

Expressões matemáticas tais como $(x_1 + x_3)/x_2$ são efetuados do seguinte modo:

```
-->(x(1)+x(3))/x(2)
 ans  =

    1.
```

Se o vetor for grande, pode-se usar o símbolo ".." para continuar escrevendo da linha seguinte. Exemplo:

```
-->b = [2 3 7 ..
-->9 8 4]
 b  =

!   2.    3.    7.    9.    8.    4. !
```

5.1.1 Acessando Elementos do Vetor

Os elementos de um vetor podem ser acessados de várias maneiras. Por exemplo, considere as variáveis:

```
a = [2 4 5 1 3];
i = 2;
```

Então tem-se:

Índices com valor zero ou negativo não são válidos no Scilab.

`a(i+2)`	←—Devolve 1 porque acessa o quarto elemento (`i+2` é igual a 4) do vetor `a`.
`a(a(4))`	←—Devolve 2. Como valor de `a(4)` é 1, avaliar `a(a(4))` é o mesmo avaliar `a(1)`. Logo, `a(a(4))` é igual a `a(1)` que, por sua vez, é igual a 2.
`a(a(3))+a(2*i)`	←—Devolve 4, porque `a(a(3))` é igual `a(5)` que, por sua vez, é igual a 3. E `a(2*i)` é igual `a(4)`, que é igual a 1. Logo, `a(a(3))+a(2*i)` é igual a 3 + 1 = 4.

Exercício resolvido 5.1.1. Calcular a média dos elementos do vetor `nota` dado na Figura 5.1.

```
1  nota = [8.1 5.2 9.2 7.2 6.5 5.2 8.5 9.5 6.5 10.0];
2  soma = 0;
3  for i=1:10
4    soma = soma + nota(i);
5  end
6  printf("Média das notas = %g\n",soma/10);
```

Comentário. Para somar os elementos do vetor, cada elemento foi acessado individualmente e adicionado, um por vez, em um acumulador `soma`, através do laço `for...end` (linhas 3 a 5).

Exercício resolvido 5.1.2. Ler dois vetores A e B de 10 elementos. Construir um vetor C tal que cada elemento de C seja o dobro da soma entre os elementos correspondentes de A com B. Escrever o vetor C.

```
1  for i=1:10                          // Leitura de A e B
2    a(i) = input("Digite um valor");
3  end
4  for i=1:10
5    b(i) = input("Digite um valor");
6  end
7  for i=1:10
8    c(i) = 2*(a(i)+b(i));             // Calculo de C
9  end
10 for i=1:10                          // Escreve de C
11   printf("%g ",c(i));
12 end
```

Comentário. O laço `for...end`, das linhas 1 a 3, faz a leitura de um elemento do vetor `a` de cada vez. A leitura é controlado pelo índice que faz com que cada leitura seja armazenado em um elemento diferente do vetor `a`. No laço `for...end` das linhas 7 a 9, o cálculo de cada elemento do vetor `c` é controlado pelo índice que faz com que seja somando os elementos correspondentes de `a` e `b`.

Exercício resolvido 5.1.3. Ler dois vetores A e B de 10 elementos. Construir um vetor C tal que o elemento de índice ímpar de C seja igual ao elemento correspondente de A, caso contrário, seja igual ao elemento correspondente de B. Por exemplo, `c[1]==a[1]`, `c[3]==a[3]`, ... Mas, `c[2]==b[2]`, `b[4]==a[4]`, ... Escrever o vetor C.

```
1  for i=1:10                          // Leitura de A e B
2    a(i) = input("Digite um valor");
3  end
4  for i=1:10
5    b(i) = input("Digite um valor");
6  end
7  for i=1:10
8    if modulo(i,2)<>0                 // Testa se o índice i é ímpar.
9       c(i) = a(i);                   // Se for ímpar c(i) recebe o valor de a(i)
10   else
11      c(i) = b(i);                   // Se for par c(i) recebe o valor de b(i)
12   end
13 end
14 for i=1:10                          // Escreve o vetor C
15   printf("%g ",c(i));
16 end
```

Comentário. A função `modulo` (resto de uma divisão) verifica se o índice do vetor C é ímpar ou par porque se resto da divisão entre um número qualquer e dois é diferente de zero então ele é ímpar (não é divisível por dois).

5.2 MATRIZES BIDIMENSIONAIS

Matrizes bidimensionais usam dois índices para individualizar elementos. Na Figura 5.2 é mostrada uma matriz. Matrizes são construídas usando colchetes. Cada linha da matriz é separada por *um ponto e vírgula* e cada elemento de uma linha é separado por espaço (ou vírgula). Por exemplo, a seguinte matriz da matemática,

$$A = \begin{bmatrix} 2 & 3 & 4 \\ 4 & 5 & 2 \end{bmatrix}$$

poderia ser construída pelo comando:

```
-->a = [2 3 4; 4 5 2]
 a  =

!   2.    3.    4. !
!   4.    5.    2. !
```

Alternativamente, mudando-se a linha (teclando enter) também separa as linhas da matriz. Exemplo:

```
-->a = [2 3 4                  // tecle enter aqui
-->     4 5 2]
 a  =

!   2.    3.    4. !
!   4.    5.    2. !
```

Os elementos são acessados com dois índices. Por exemplo,

```
-->a(1,2)
 ans  =

    3.
```

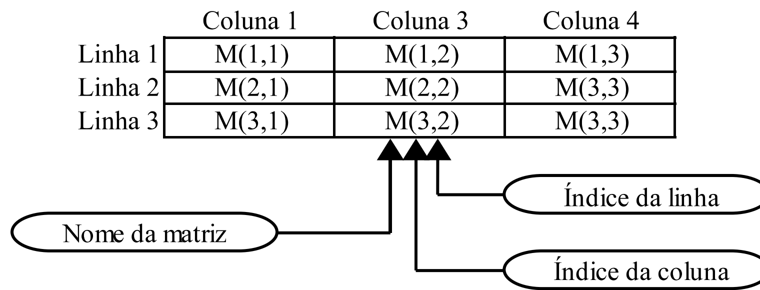


Figura 5.2: Matriz bidimensional

```
-->a(2,3)
 ans  =

    2.

-->a(1,3)
 ans  =

    4.
```

Exercício resolvido 5.2.1. Dado a matriz A:

$$A = \begin{bmatrix} 3 & 1 & 2 & 4 \\ 5 & 5 & 8 & 6 \\ 8 & 10 & 11 & 5 \\ 9 & 1 & 5 & 7 \\ 2 & 3 & 8 & 8 \end{bmatrix}$$

Pede-se:

a) Colocar a matriz A na memória do computador.

```
a = [3 1 2 4; 5 5 8 6; 8 10 11 5; 9 1 5 7; 2 3 8 8];
```

b) Preencher a terceira coluna da matriz A com o valor zero.

```
a = [3 1 2 4; 5 5 8 6; 8 10 11 5; 9 1 5 7; 2 3 8 8];
for i=1:5
  a(i,3) = 0
end
```

c) Calcular a soma dos elementos da diagonal principal da matriz A e escrever o resultado.

```
a = [3 1 2 4; 5 5 8 6; 8 10 11 5; 9 1 5 7; 2 3 8 8];
soma = 0;
for i=1:4
  soma = soma + a(i,i);
end
printf("soma = %g\n",soma);
```

d) Calcular a soma dos quadrados dos elementos da segunda linha de A e escrever o resultado.

```
a = [3 1 2 4; 5 5 8 6; 8 10 11 5; 9 1 5 7; 2 3 8 8];
soma = 0;
for i=1:4
  soma = soma + a(2,i)^2;
end
printf("soma = %g\n",soma);
```

e) Somar de todos os elementos de A.

```
a = [3 1 2 4; 5 5 8 6; 8 10 11 5; 9 1 5 7; 2 3 8 8];
soma = 0;
for i=1:5
  for j=1:4
    soma = soma + a(i,j);
  end
end
```

f) Armazenar a soma de cada linha de A no vetor S (ver Figura 5.2)

```
a = [3 1 2 4; 5 5 8 6; 8 10 11 5; 9 1 5 7; 2 3 8 8];
for i=1:5
  soma = 0;
  for j=1:4
    soma = soma + a(i,j);
  end
  s(i) = soma;
end
```

g) Trocar a segunda linha com quarta linha.

```
a = [3 1 2 4; 5 5 8 6; 8 10 11 5; 9 1 5 7; 2 3 8 8];
for j=1:4
  aux = a(2,j)
  a(2,j) = a(4,j);
  a(4,j) = aux;
end
```

h) Escrever a matriz A.

```
a = [3 1 2 4; 5 5 8 6; 8 10 11 5; 9 1 5 7; 2 3 8 8];
for i=1:5
  for j=1:4
    printf("%3.0f ",a(i,j));
  end
  printf("\n");
end
```

i) Ler uma matriz B de mesma dimensão que A. Efetuar a soma matricial $A + B$ e armazenar o resultado na matriz C.

```
a = [3 1 2 4; 5 5 8 6; 8 10 11 5; 9 1 5 7; 2 3 8 8];
for i=1:5
  for j=1:4
    b(i,j) = input("Digite um número");
  end
end
for i=1:5
  for j=1:4
```

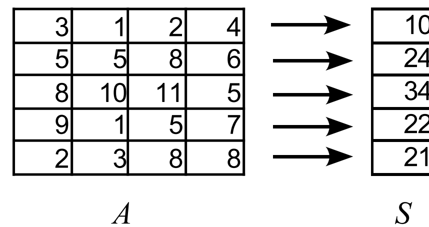


Figura 5.3: Soma das linhas da matriz A.

```
    c(i,j) = a(i,j) + b(i,j);
  end
end
```

j) Ler uma matriz B de mesma dimensão que A. Escrever IGUAIS se A for uma matriz igual B. Caso contrário, escrever DIFERENTES.

```
a = [3 1 2 4; 5 5 8 6; 8 10 11 5; 9 1 5 7; 2 3 8 8];
for i=1:5
  for j=1:4
    b(i,j) = input("Digite um número");
  end
end
iguais = %t;
for i=1:5
  for j=1:4
    if a(i,j) <> b(i,j)
      iguais = %f;
    end
  end
end
if iguais
  printf("Iguais\n");
else
  printf("Diferentes\n");
end
```

Exercício 5.2.2. Elabore um programa para ler a matriz A, trocar a segunda linha com terceira linha e escrever a matriz A.

5.3 VETORES DE STRING

Podemos construir vetores de strings. Por exemplo:

```
-->mes = ["jan" "fev" "mar" "abr" "jun" "jul" "ago" "set" "nov" "dez"];
```

O vetor `mes` foi construído de tal modo que há uma correspondência entre o número do mês e o índice do elemento. Por exemplo, o mês de número 11 (novembro) é acessado pelo elemento mes(11). Este fato é usado para resolver o exercício a seguir.

Exercício resolvido 5.3.1. Elaborar um programa que leia o dia, o número de mês e o ano e escreva a dada no formato “D de MMM de AAAA”(ou “D de MMM de AA”). Por exemplo, se o dia é 31, o mês é 12 e o ano é 2003, então o programa deverá escrever `1 de dez de 2003`.

Solução:

```
mes = ["jan" "fev" "mar" "abr" "jun" "jul" "ago" "set" "nov" "dez"];
dia = input("Digite o dia");
nunmes = input("Digite o numero do mes");
ano = input("Digite o ano");
printf("%g de %s de %g\n",dia,mes(nunmes),ano);
```

Resultado

```
Digite o dia-->2
Digite o numero do mes-->4
Digite o ano-->2003
2 de abr de 2003
```

Comentário. Porque a variável `nunmes` contém o número do mês, o comando `printf` é capaz de escrever o nome do mês através do elemento `mes(nunmes)`.

Exercícios

O que este programa escreve?

```
poema(1) = 'uma rosa';
poema(2) = 'é';
for a=1:3
   for b=1:2
      printf("%s ",poema(b));
   end
end
printf('%s',poema(1));
```

5.4 ESTUDO DE CASO

Esta seção ilustra a facilidade que as matrizes oferecem a programação de computadores. Primeiramente, será mostrado a solução de um problema sem o uso de vetores visando apontar as dificuldades deste procedimento. Em seguida, o mesmo problema será resolvido com o uso de vetores. Considere o problema:

> *Suponha uma turma com quatro alunos. Elaborar um programa que leia as quatro notas dos alunos e seus respectivos nomes e escreva apenas os nomes com a nota acima da média.*

Neste sentido, uma possível solução (sem vetores) seria a seguinte:[2]

[2] Comumente o cálculo da média das notas pode ser efetuado pelo seguinte trecho de programa:

```
soma = 0;
for i = 1:4
  nota = input("Digite a nota");
  soma = soma + nota;
end
media = soma/4;
```

Esta abordagem não resolve o problema proposto porque precisamos comparar se cada nota digitada é maior que a média. Mas vale notar que as notas já foram perdidas no momento que é calculado a média na última linha (exceto a última nota que não foi perdida porque está armazenada na variável nota). Deste modo, não podemos comparar as média com as notas (que foram perdidas). Portanto, a fim de evitar a perda das notas, é necessário armazenar cada nota digitada em uma variável diferente.

```
aluno1  = input("Nome do aluno","s");
nota1   = input("Nota");
aluno2  = input("Nome do aluno","s");
nota2   = input("Nota");
aluno3  = input("Nome do aluno","s");
nota3   = input("Nota");
aluno4  = input("Nome do aluno","s");
nota4   = input("Nota");
media = (nota1+nota2+nota3+nota4)/4;
if nota1 >= media
  printf("%s\n",aluno1);
end
if nota2 >= media
  printf("%s\n",aluno2);
end
if nota3 >= media
  printf("%s\n",aluno3);
end
if nota4 >= media
  printf("%s\n",aluno4);
end
```

Suponha que, ao invés de uma turma com quatro alunos, houvessem uma turma com cinco alunos, então teríamos:

> *Suponha uma turma com cinco alunos. Elaborar um programa que leia as cinco notas dos alunos e seus respectivos nomes e escreva apenas os nomes com a nota acima da média.*

A solução deste problema seria similar ao programa anterior, apenas acrescentando algumas linhas (e.g., para ler a quinta nota). A solução seria:

```
aluno1  = input("Nome do aluno","s");
nota1   = input("Nota");
aluno2  = input("Nome do aluno","s");
nota2   = input("Nota");
aluno3  = input("Nome do aluno","s");
nota3   = input("Nota");
aluno4  = input("Nome do aluno","s");
nota4   = input("Nota");
aluno5  = input("Nome do aluno","s");
nota5   = input("Nota");
media = (nota1+nota2+nota3+nota4+nota5)/5;
printf("Relação de alunos acima da média\n");
if nota1 >= media
  printf("%s\n",aluno1);
end
if nota2 >= media
  printf("%s\n",aluno2);
end
if nota3 >= media
  printf("%s\n",aluno3);
end
if nota4 >= media
  printf("%s\n",aluno4);
end
if nota5 >= media
```

```
26    printf("%s\n",aluno5);
27  end
```

Novamente, suponha que, ao invés de uma turma com cinco alunos, houvessem uma turma com dez alunos. Neste caso, um programa, para ler 10 notas, iria crescer bastante em tamanho (em relação programa anterior) porque seria preciso ler 10 variáveis para as notas (i.e., `nota1`, `nota2`, ..., `nota10`) e 10 para os nomes de alunos, totalizando 20 variáveis. Do mesmo modo, para uma turma com 20 alunos, seria preciso ler 40 variáveis, sendo as 20 notas e os 20 nomes dos alunos.

Considere a situação que fosse preciso ler as notas de todas as turmas de um colégio com 10000 alunos. Ou seja,

> *Suponha uma colégio com 10000 alunos. Elaborar um programa que leia as 10000 notas dos alunos e seus respectivos nomes e escreva apenas os nomes com a nota acima da média.*

Este programa assume proporção gigantesca porque é preciso ler 20000 variáveis (10000 para as notas e 10000 para os nomes de aluno) tornando-o impraticável de ser programado (do modo como foi feito antes). Note que as variáveis,

`nota1`, `nota2`, `nota3`, ...

podem ser substituídas por os elementos de um vetor, por exemplo,

`nota(1)`, `nota(2)`, `nota(3)`, ...

Neste caso, cada elemento do vetor nota seria lida, uma de cada vez, por um laço `for...end` através dos seus índices. Vejamos a solução usando vetores.

```
1   soma = 0;   // acumulador das notas
2   for i = 1:10000
3     aluno(i) = input("Digite o nome do aluno","s");
4     nota(i) = input("Nota");
5     soma = soma + nota(i);
6   end
7   media = soma/10000;   // calculo da media
8   for i = 1:10000
9     if nota(i) >= media
10      printf("%s\n",aluno(i));
11    end
12  end
```

Comentário. No laço `for...end` das linhas 3 a 6, as notas e os nomes de aluno são lido um de cada vez. O acumulador `soma`, dentro do laço, soma todas as notas (uma por vez). Os nomes dos alunos com nota acima da média são escritos no segundo laço `for...end` nas linha 8 a 12. Cada nota é comparada, uma de cada vez, com a média e caso a nota de índice i passe no teste, o correspondente nome do aluno de índice i é escrito.

A conclusão é que o uso de vetores permite ler e armazenar grande quantidade de notas (e nomes) em um simples laço `for...end` reduzindo, deste modo, o tamanho de um programa cuja elaboração sem o uso de vetores se torna impraticável.

5.5 EXEMPLOS COM MATRIZES

5.5.1 Ordenação de Vetores

A ordenação de elementos de um conjunto é fundamental no dia a dia. Um exemplo bem comum são as listas telefônicas cuja ordenação em ordem alfabética possibilita rapidamente encontrar um número de telefone. Um outro exemplo é a ordenação de livros por autor ou por assunto em uma grande biblioteca. Por isso a ordenação é uma atividade constantemente solicitada em programação. Provavelmente, o algoritmo de ordenação mais conhecido da computação chama-se o **algoritmo da bolha**.

A idéia do algoritmo da bolha é passar por todo o vetor comparando cada elemento com o elemento seguinte (i.e., a(i) com a(i+1)) e troca-los caso estes dois elementos não estejam na ordem apropriada. Por diversas vezes, o algoritmo repete a passada até o vetor ficar completamente ordenado.

Vejamos como funciona este algoritmo para ordenar o `a = [5 4 3 2 6]`.

A) Vetor inicial desordenado: `a = [5 4 3 2 6]`

Passo A.1 – Comparar a(1) e a(2). Se a(1) > a(2), troque os dois elementos.
`a = [4 5 3 2 6]`
Houve troca do número 4 com o número 5.

Passo A.2 – Comparar a(2) e a(3). Se a(2) > a(3), troque os dois elementos.
`a = [4 3 5 2 6]`
Houve troca do número 5 com o número 3.

Passo A.3 – Comparar a(3) e a(4). Se a(3) > a(4), troque os dois elementos.
`a = [4 3 2 5 6]`
Houve troca.

Passo A.4 – Comparar a(4) e a(5). Se a(4) > a(5), troque os dois elementos.
`a = [4 3 2 5 6]`
Não houve troca porque a(4) não é maior que a(5).

B) Repetição a iteração (porque o vetor ainda não está ordenado):

Passo B.1 – Comparar a[1] e a[2]. Se a[1] > a[2], troque os dois elementos.
`a = [3 4 2 5 6]`
Houve troca do número 4 com o número 3.

Passo B.2 – Comparar a[2] e a[3]. Se a[2] > a[3], troque os dois elementos.
`a = [3 2 4 5 6]`
Houve troca do número 4 com o número 2.

Passo B.3 – Comparar a[3] e a[4]. Se a[3] > a[4], troque os dois elementos.
`a = [3 2 4 5 6]`
Não houve troca.

Passo B.4 – Comparar a[4] e a[5]. Se a[4] > a[5], troque os dois elementos.
`a = [3 2 4 5 6]`
Novamente não houve troca.

C) Repetição a iteração (porque o vetor não está ordenado)

Passo C.1 – Comparar a[1] e a[2]. Se a[1] > a[2], troque os dois elementos.
`a = [2 3 4 5 6]`
Houve troca do número 3 com o número 2.

Passo C.2 – Comparar a[2] e a[3]. Se a[2] > a[3], troque os dois elementos.
`a = [2 3 4 5 6]`
Não houve troca.

Passo C.3 – Comparar a[3] e a[4]. Se a[3] > a[4], troque os dois elementos.
`a = [2 3 4 5 6]`
Não houve troca.

Passo C.4 – Comparar a[4] e a[5]. Se a[4] > a[5], troque os dois elementos.
`a = [2 3 4 5 6]`
Não houve troca.

D) Repetição a iteração.

(a) **Passo D.1** – Comparar a[1] e a[2]. Se a[1] > a[2], troque os dois elementos.
`a = [2 3 4 5 6]`
Não houve troca.

(b) **Passo D.2** – Comparar a[2] e a[3]. Se a[2] > a[3], troque os dois elementos.
`a = [2 3 4 5 6]`
Não houve troca.

(c) **Passo D.3** – Comparar a[3] e a[4]. Se a[3] > a[4], troque os dois elementos.
`a = [2 3 4 5 6]`
Não houve troca.

(d) **Passo D.4** – Comparar a[4] e a[5]. Se a[4] > a[5], troque os elementos.
`a = [2 3 4 5 6]`
Não houve troca.

E) Se não houve troca na última iteração então pare porque o vetor já está ordenado.

O seguinte trecho de programa implementa o algoritmo da bolha para ordenar a vetor `a` (supondo que o vetor `a` já foi lido e está na memória).

```
1  n = length(a);            // Obtem o tamanho do vetor
2  HouveTroca = %t;
3  while HouveTroca
4    HouveTroca = %f;
5    for i = 1:(n-1)
6      if a(i) > a(i+1)
7        temp = a(i);        // troca o elemento i com i+1
8        a(i) = a(i+1);
9        a(i+1) = temp;
10       HouveTroca = %t;    // Houve troca
11     end
12   end
13 end
```

Comentário. A variável booleana `HouveTroca` indica se houve troca ou não após a última passada pelo vetor. A variável `HouveTroca` assume o valor falso na linha 4, e se dentro do laço interno `for...end` não houver troca então a variável HouveTroca continuará com valor falso (isto fará com que o laço `while...end` encerre a execução do programa). Caso contrário, a variável `HouveTroca` assume o valor verdadeiro na linha 10) (isto fará com que o laço `while...end` execute nova iteração).

A troca de um elemento com o elemento seguinte (i.e., a(i) com a(i+1)) é realizado pelo trecho:

```
temp = a(i);
a(i) = a(i+1);
a(i+1) = temp;
```

A variável `temp` é uma variável temporária usada apenas para guardar o valor de `a(i)` antes que o mesmo seja perdido na atribuição `a(i) = a(i+1)`.

Exercício 5.5.1. Modifique trecho de programa que contém algoritmo da bolha para colocar o vetor em ordem decrescente (troque o operador > por < na linha 6). Use esta modificação para elaborar um programa para ler um vetor com 5 elementos, coloca-lo em ordem decrescente e escreve-lo.

5.5.2 Gerando Números Aleatórios

Exercício resolvido 5.5.2. Elabore um programa que, simulando lançamentos de dados, calcule a freqüência de cada resultado após 100 lançamentos.

Para resolver este problema utiliza-se a função `rand` (também chamada de *gerador de números aleatórios*). Esta função retorna um número diferente (aleatório) cada vez que é chamada, por exemplo:

```
-->rand()
 ans  =

     .4410204

-->rand()
 ans  =

     .8859080

-->rand()
 ans  =

     .6868068
```

Os números aleatórios gerados por `rand` são números fracionários no intervalo [0, 1]. No entanto, o problema pede para gerar números inteiros aleatoriamente entre 1 e 6 de modo a simular o lançamento de um dado. Isto pode ser obtido com o seguinte procedimento. Primeiro, multiplica-se `rand` por 6 para gerar números no intervalo $[0, 6]$. Por exemplo:

```
-->6*rand()
```

Em seguida, usa-se a função `fix()` para gerar apenas números inteiros entre 0 e 5 (inclusive). Por exemplo,

```
-->fix(6*rand())
```

Por fim, adiciona-se o valor 1 na expressão anterior para gerar números inteiros entre 1 e 6.

```
-->fix(6*rand())+1)
```

e obtém-se, deste modo, números inteiros aleatórios entre 1 e 6 na qual simula o lançamento de um dado como requerido.

Será utilizado, também, um vetor `f` que armazena a freqüência de cada dado. Por exemplo, o elemento f(1) armazena a freqüência do dado 1, o elemento f(2) armazena a freqüência do dado 2, e assim por diante. Solução:

```
1  for i = 1:6
2    f(i) = 0;     // inicializa o vetor de frequencias
3  end
4  for i=1:100
5    r = fix(6*rand())+1;  // lançamento do dado
6    f(r) =  f(r)+1;       // adiciona 1 ao dado r
7  end
8  for i=1:6
9    f(i) =  f(i)/100;     // divide o vetor f por 100 para obter a frequencia.
10   printf("freqüência do dado %1.0f = %5.2f\%\n",i,f(i));
11 end
```

Uma possível execução do programa seria:

```
Resultado
freqüência do dado 1 =  0.12%
freqüência do dado 2 =  0.13%
freqüência do dado 3 =  0.29%
freqüência do dado 4 =  0.22%
freqüência do dado 5 =  0.13%
freqüência do dado 6 =  0.11%
```

5.5.3 Uma Aplicação de Matrizes

Exercício resolvido 5.5.3. Alguns candidatos prestaram concurso para cinco vagas em uma empresa. Os resultados das provas do concurso estão armazenadas nas seguintes matrizes:

Nome
Ana
Carlos
Francisco
José
Magali
Marcos
Maria
Marta
Paulo
Pedro

Matemática	Português	Digitação
6,5	7,1	7,5
8,0	8,3	7,2
7,5	8,1	8,3
6,1	5,2	6,0
5,1	6,1	6,5
4,1	5,5	5,4
9,1	8,9	9,4
8,8	8,5	9,0
9,5	9,3	9,1
8,2	8,5	7,8

Escreva um programa que:

a) Armazene o nomes dos candidados em um vetor nome e as notas das provas em uma matriz nota 10×3.

   O símbolo ... será usado para continuar os comandos na linha seguinte.

```
nome = ["Ana" "Carlos" "Francisco" ...
        "José" "Magali" "Marcos" ...
         "Maria" "Marta" "Paulo" "Pedro"];

nota = [6.5 7.1 7.5; ...
        8.0 8.3 7.2; ...
        7.5 8.1 8.3; ...
        6.1 5.2 6.0; ...
```

```
        5.1 6.1 6.5; ...
        4.1 5.5 5.4; ...
        9.1 8.9 9.4; ...
        8.8 8.5 9.0; ...
        9.5 9.3 9.1; ...
        8.2 8.5 7.8];
```

b) Calcule a média de cada candidato, armazene em um vetor media e escreva-o.

Neste programa e nos seguintes é assumido que o código acima já está incluído no programa.

```
1  [m,n]=size(nota);
2  for i=1:m
3    soma = 0;
4    for j=1:n
5      soma = soma+nota(i,j);
6    end
7    media(i)=soma/3;
8  end
9  printf("Nome       Media\n");   // linha de cabeçario
10 for i=1:m
11   printf("%-10s %3.1f\n",nome(i),media(i));
12 end
```

Resultado

```
Nome       Media
Ana        7.0
Carlos     7.8
Francisco  8.0
José       5.8
Magali     5.9
Marcos     5.0
Maria      9.1
Marta      8.8
Paulo      9.3
Pedro      8.2
```

c) Calcule e escreva a maior nota da prova de português e a nome do respectivo candidato.

```
1  maior = 0.0;
2  m = size(nota,1);
3  for i=1:m
4    if nota(i,2) > maior
5      maior = nota(i,2);
6      imaior = i;         // Armazena o índice da maior nota.
7    end
8  end
9  printf("A maior nota de português: %s, %3.1f\n",nome(imaior),maior);
```

d) Elabore um programa para escrever um relatório dos candidatos em ordem de classificação dos candidados.

O solução desta questão é mostrado no programa listado abaixo. Na primeira parte do programa (linhas 1-8), calcula-se a média da mesma maneira que no item(b) deste exercício. Na segunda parte do programa (linhas 10-25) é feita a ordenação dos vetores com o algoritmo da bolha. O vetor ordenado é o vetor `media`, mas

note que para cada dois elementos trocados do vetor do `media` é também trocado os respectivos elementos do vetor `nome` para acompanhar a mesma ordem do vetor `media`. Na última parte do programa (linha 27-30) é impresso o relatório dos candidatos.

```
1   [m,n]=size(nota);
2   for i=1:m               // Computa as medias dos alunos
3     soma = 0;
4     for j=1:n
5       soma = soma+nota(i,j);
6     end
7     media(i)=soma/3;
8   end
9
10  // ordenação e troca dos elementos
11  HouveTroca = %t;
12  while HouveTroca
13    HouveTroca = %f;
14    for i = 1:(m-1)
15      if media(i) < media(i+1)
16        temp = media(i);       // troca o elemento media i com i+1
17        media(i) = media(i+1);
18        media(i+1) = temp;
19        temp = nome(i);       // troca o elemento nome2 i com i+1
20        nome(i) = nome(i+1);
21        nome(i+1) = temp;
22        HouveTroca = %t;   // Houve troca
23      end
24    end
25  end
26
27  printf("    Nome       Media\n");   // linha de cabeçario
28  for i=1:m
29    printf("%2.0f- %-10s %3.1f\n",i,nome2(i),media(i));
30  end
```

Resultado

```
    Nome       Media
 1- Paulo      9.3
 2- Maria      9.1
 3- Marta      8.8
 4- Pedro      8.2
 5- Francisco  8.0
 6- Carlos     7.8
 7- Ana        7.0
 8- Magali     5.9
 9- José       5.8
10- Marcos     5.0
```

Capítulo 6

MANIPULAÇÃO MATRICIAL

A manipulação matricial é uma das mais interessantes características do Scilab porque reduzem *susbtancialmente* a quantidade de linhas de código e, frequentemente, torna o programa mais eficiente.

6.1 CONSTRUÇÃO DE MATRIZES

O Operador Dois Pontos

O operador dois pontos é usado para construir uma sequência de valores. Este operador define o valor inicial e final da sequência e o incremento entre os valores. O operador dois pontos tem a seguinte forma:

```
inicio:incremento:fim
```

Exemplo,

```
-->1:2:15
 ans  =

!   1.    3.    5.    7.    9.    11.    13.    15. !
```

O incremento assume o valor de um se ele for omitido:

```
-->1:5
 ans  =

!   1.    2.    3.    4.    5. !
```

Note que o operador dois pontos é também usado no comando FOR.

Funções Aplicadas a Matrizes

Muitas funções comuns (e.g., `sin(x)`, `cos(x)`, `abs(x)`, etc) são definidas para receber valores e retornar valores. Mas quando recebem matrizes, estas funções operam elemento por elemento na matriz. Por exemplo, calcularemos o logaritmo natural de números entre 10 e 20 separados por intervalos de 2,5:

```
-->a=10:2.5:20
 a  =

!   10.    12.5    15.    17.5    20. !

-->log(a)       // a função é aplicada a cada elemento
```

```
 ans  =

!   2.3025851    2.5257286    2.7080502    2.8622009    2.9957323 !
```

Calculo do seno entre 0 e π com incremento de $\pi/4$:

```
-->sin(0:%pi/4:%pi)
 ans  =

!   0.    0.7071068    1.    0.7071068    1.225E-16 !
```

Transposta de uma Matriz

O operador de transposição (’) constroi a transposta de uma matriz:

```
-->a = [1; 2; 3]            // Um vetor coluna
 a  =

!   1. !
!   2. !
!   3. !

-->a’                       // A transposta gera um vetor linha
 ans  =

!   1.    2.    3. !
```

Concatenação de Matrizes

Uma operação muito comum é a concatenação de matrizes. Por exemplo:

```
-->a = [1 2 3];

-->b = [4 5 6];

-->[a b]
 ans  =

!   1.    2.    3.    4.    5.    6. !

-->[a; b]
 ans  =

!   1.    2.    3. !
!   4.    5.    6. !

->[a; 5 4 3]
 ans  =

!   1.    2.    3. !
!   5.    4.    3. !
```

Mais exemplos:

```
-->c = [1 2 3 4];

-->d = [2 3 1 2; 3 4 5 4; 5 6 7 4]
 d  =
```

```
    2.    3.    1.    2.
    3.    4.    5.    4.
    5.    6.    7.    4.

-->[d; c; c]
 ans  =

    2.    3.    1.    2.
    3.    4.    5.    4.
    5.    6.    7.    4.
    1.    2.    3.    4.
    1.    2.    3.    4.

-->[d a' b']
 ans  =

    2.    3.    1.    2.    1.    4.
    3.    4.    5.    4.    2.    5.
    5.    6.    7.    4.    3.    6.
```

Este programa produz um vetor de comprimento n:

```
1  n = 10;
2  x = [];    // x é inicializado com uma matriz vazia
3  for i=n:-1:1
4     x = [x i^2];
5  end
```

Vale resaltar que a concatenação só é possível se houver *consistência dimensional* entre as matrizes. Por exemplo, o seguinte exemplo gera um erro por que o vetor c tem dimensões inconsistentes com os vetores a e b:

```
-->a = [1 2 3];

-->b = [4 5 6];

-->c = [1 2 3 4];

-->[a; b; c]
         !--error     6
inconsistent row/column dimensions
```

linspace e logspace

Construímos um vetor de 5 elementos igualmente espaços entre 0 e 10 com o operador dois pontos da seguinte forma:

```
-->1:0.125:1.5
 ans  =

!   1.    1.125    1.25    1.375    1.5 !
```

Note que precisamos conhecer o incremento de 0,125 para gerar o vetor acima. Se o incremento não é conhecido, pode ser mais simples usar a função `linspace` para gerar o vetor acima. Sua sintaxe é

```
linspace(inicio,fim,n)
```

Tabela 6.1: Construção de Matrizes

`x = inicio:fim`	Cria um vetor linha que começa em `inicio`, incrementa em um e um até atingir `fim`.
`x = inicio:incremento:fim`	Cria um vetor linha que começa em `inicio`, incrementando de `incremento` até atingir `fim`.
`x = linspace(inicio,fim,n)`	Cria um vetor linha com n valores igualmente espaçados começando em `inicio` e terminando em `fim`.
`x = logspace(inicio,fim,n)`	Cria um vetor linha com n valores logaritmicamente espaçados começando em `inicio` e terminando em `fim`.

`a = [1 2 3];`
`b = [8 9 2];`
`x = [a b];`
	Cria vetor linha com os elementos de `a` e `b`.

`x = [a; b];`	Cria matriz com os elementos de `a` na primeira linha e os elementos de `b` na segunda linha.

A função `linspace` gera um vetor com n valores igualmente espaçados começando em `inicio` e terminando em `fim`. Exemplo:

```
-->linspace(1,1.5,5)
 ans  =

!   1.    1.125    1.25    1.375    1.5 !
```

A função `logspace(inicio,fim,n)` gera n elementos logaritmicamente espaçados. O primeiro elemento é 10 elevado ao `inicio` e o último elemento é 10 elevado ao `fim`. Por exemplo:

```
-->logspace(1,2,4)
 ans  =

!   10.    21.544347    46.415888    100. !
```

O seguinte exemplo plota o gráfico da função seno. A função `plot2d(x,y)` cria um gráfico onde `x` e `y` são vetores de mesmo tamanho representando os pontos do eixo x e y de uma função qualquer. Usamos 50 pontos no intervalo de 0 até 2π:

```
-->x = linspace(0,2*%pi,50);

-->y = sin(x);

-->plot2d(x,y);
```

Na verdade, a função `plot2d(x,y)` apenas conecta os pontos dados por meio de linhas retas. Portanto, para obter uma aparência suave da curva do gráfico é preciso uma quantidade suficiente de pontos (no caso usamos 50 pontos).

Um resumo das operações construção de matrizes é mostrado na Tabela 6.1.

6.2 SECIONAMENTO DE MATRIZES

O Scilab permite manipulação de parte da matriz (uma submatriz). Esta operação é denominada de secionamento (do inglês, *slicing*). Considere o vetor:

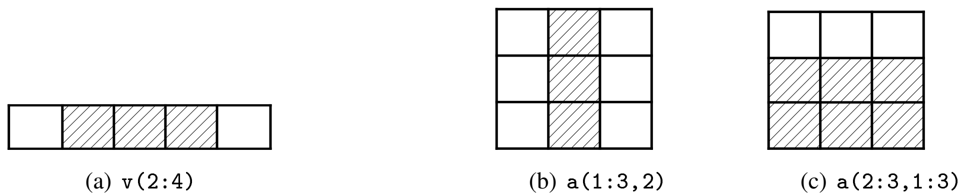


Figura 6.1: Secionamento de Matrizes

```
-->v = [2 5 6 3 8]
 v  =

!   2.    5.    6.    3.    8. !
```

Obtemos uma parte do vetor (subvetor) usando o comando `v(2:4)` que refere-se aos elementos 2, 3, e 4 do vetor `v` (ver Figura 6.1(a)):

```
-->v(2:4)
 ans  =

!   5.    6.    3. !
```

Considere a matriz:

```
-->a = [2 3 1; 7 8 4; 2 8 8]
 a  =

!   2.    3.    1. !
!   7.    8.    4. !
!   2.    8.    8. !
```

`a(1:3,2)` refere-se a segunda coluna da matriz `a` (ver Figura 6.1(b)):

```
-->a(1:3,2)
 ans  =

!   3. !
!   8. !
!   8. !
```

`a(2:3,1:3)` refere-se as linhas 2 e 3 da matriz `a` (ver Figura 6.1(c)):

```
-->a(2:3,1:3)
 ans  =

!   7.    8.    4. !
!   2.    8.    8. !
```

O operador dois pontos sem argumentos refere-se ao intervalo inteiro:

```
-->a(:,1)      // refere-se a primeira coluna
 ans  =

!   2. !
!   7. !
!   2. !

-->a(2,:)      // refere-se a segunda linha
 ans  =

!   7.    8.    4. !
```

Para extrair a primeira e terceira linhas usamos:

```
-->a([1 3],:)
 ans  =

!   2.    3.    1. !
!   2.    8.    8. !
```

Note que este exemplo troca a coluna 1 da matriz pela coluna 3:

```
-->a(:,3:-1:1)
 ans  =

!   1.    3.    2. !
!   4.    8.    7. !
!   8.    8.    2. !
```

Vetores pode ser usados como índice. Por exemplo,

```
-->x=[1 2 3];

-->a(2,x)
 ans  =

!   7.    8.    4. !
```

Este exemplo extrai uma submatriz da matriz a:

```
-->c = [1 2];

-->a(c,c)
 ans  =

!   2.    3. !
!   7.    8. !
```

Para lembrar, mostramos a matriz a novamente:

```
-->a
 a  =

!   2.    3.    1. !
!   7.    8.    4. !
!   2.    8.    8. !
```

e um exemplo de concatenação de matriz:

```
-->[a ; a(1:2,:)]
 ans  =

!   2.    3.    1. !
!   7.    8.    4. !
!   2.    8.    8. !
!   2.    3.    1. !
!   7.    8.    4. !
```

Apesar de não ser a maneira mais simples, a transposta da matriz a pode ser calculada da seguinte forma:

```
-->[a(:,1)';a(:,2)';a(:,3)']
 ans  =

!   2.    7.    2. !
!   3.    8.    8. !
!   1.    4.    8. !
```

2 1	3 5	1 9
7 2	8 6	4 10
2 3	8 7	8 11
6 4	4 8	5 12

índice linear

Figura 6.2: Indexação Linear

6.2.1 Indexação Linear

A operação de indexar, com apenas um índice, uma matriz bidimensional é chamada de *indexação linear*. A matriz é tratada como se fosse um longo vetor coluna formado pelas colunas da matriz (uma coluna abaixo da outra). Por exemplo, a Figura 6.2.1 mostra índices lineares da seguinte matriz:

```
-->A = [2 3 1; 7 8 4; 2 8 8; 6 4 5]
 A  =

!   2.    3.    1. !
!   7.    8.    4. !
!   2.    8.    8. !
!   6.    4.    5. !
```

A operação `A(i)` retorna elemento de A com o i-ésimo índice linear. Exemplo:

```
-->A(8)
 ans  =

    4.
```

A operação `A(:)` retorna um vetor coluna construído pelas colunas da matriz `A`:

```
-->A(:)
 ans  =

!   2. !
!   7. !
!   2. !
!   6. !
!   3. !
!   8. !
!   8. !
!   4. !
!   1. !
!   4. !
!   8. !
!   5. !
```

Tabela 6.2: Secionamento de Matrizes

Considere o vetor `v` de tamanho `n`.	
`A(v,:)`	Extrai as linhas `v(1)`, `v(2)`, `...`, `v(n)` de A.
`A(:,v)`	Extrai as colunas `v(1)`, `v(2)`, `...`, `v(n)` de A.
`A(:)`	Retorna um vetor coluna construído percorrendo as colunas da matriz `A` pela ordem crescente dos índices da coluna.
`A(v)`	Extrai os elementos de `A` cujos índices correspondem a v(1), v(2), ..., v(n), como se `A` fosse o vetor-coluna `A(:)`

Outros exemplos,

```
-->A(2:5)
 ans  =

!   7. !
!   2. !
!   6. !
!   3. !

-->A([2 8 1])
 ans  =

!   7. !
!   4. !
!   2. !
```

Um resumo de algumas operações de secionamento de matrizes é mostrado na Tabela 6.2.

6.3 O OPERADOR $

O operador $ representa o último elemento do vetor. Por exemplo, considere o vetor:

```
-->x = 10:-2:2
 x  =

!   10.    8.    6.    4.    2. !

-->x(3:$)   // extrai do terceiro elemento ao último elemento.
 ans  =

!   6.    4.    2. !
```

O operador $ pode ser utilizado para diversas operações:

```
-->x($-1)     // extrai o penúltimo elemento
 ans  =

    4.
```

```
-->x(1:2:$)   // extrai os elementos com índice impar
 ans  =

!   10.    6.    2. !

-->x($:-1:1)  // extrai os elementos na ordem inversa
 ans  =

!   2.    4.    6.    8.    10. !
```

6.4 ATRIBUIÇÃO

Podemos atribuir matrizes para um bloco de outra matriz. Considere a matriz:

```
-->A = [2 3 1; 7 4 5; 2 1 8]
 A  =

!   2.    3.    1. !
!   7.    4.    5. !
!   2.    1.    8. !
```

O seguinte exemplo altera toda a segunda linha de A:

```
-->A(2,:) = [8 7 2]
 A  =

!   2.    3.    1. !
!   8.    7.    2. !
!   2.    1.    8. !
```

Outro exemplo:

```
-->A(:,1) = 4        // O número 4 é expandido para preencher toda a coluna 1.
 A  =

!   4.    3.    1. !
!   4.    7.    2. !
!   4.    1.    8. !
```

Considere o vetor:

```
-->v = [2 4 7 1 3];

-->v([2 4]) = 8       // atribui 8 para o segundo e o quarto elemento de v
 v  =

!   2.    8.    7.    8.    3. !
```

A seguir atribuimos uma matriz 2x2 para um bloco 2x2 da matriz A:

```
-->A(2:3,1:2) = [8 9; 1 2]
 A  =

!   2.    3.    1. !
!   8.    9.    5. !
!   1.    2.    8. !
```

Quando você atribui uma matriz vazia `[]` a uma linha (ou coluna), ela é eliminada. Por exemplo,

```
-->A(2,:) = []     // A segunda linha da matriz A será eliminada
 A  =

!   2.    3.    1. !
!   1.    2.    8. !
```

Considere a matriz `B`:

```
-->B = [2 3; 5 8]
 B  =

!   2.    3. !
!   5.    8. !
```

Considere também esta atribuição:

```
-->B(3,4) = 4
 B  =

!   2.    3.    0.    0. !
!   5.    8.    0.    0. !
!   0.    0.    0.    4. !
```

Como a matriz B não possui a terceira linha e nem a quarta coluna, ela foi ampliada com os novos elementos assumindo valor zero.

6.5 DIMENSÃO DE MATRIZES

A função `length()` retorna o tamanho de um vetor. Por exemplo,

```
-->v = -%pi:%pi/2:%pi
 v  =

! - 3.1415927  - 1.5707963    0.    1.5707963    3.1415927 !

-->length(v)
 ans  =

    5.
```

A função `size()` retorna um vetor de dois elementos com o número de linhas e colunas de uma matriz. Por exemplo:

```
-->a = [1 2 3 4 5; 3 4 5 6 5]
 a  =

!   1.    2.    3.    4.    5. !
!   3.    4.    5.    6.    5. !

-->size(a)
 ans  =

!   2.    5. !
```

Neste exemplo as dimensões são armazenadas nas variáveis `l` e `c`:

```
-->[l c] = size(a)
 c  =

    5.
 l  =

    2.
```

As funções `size(x,1)` e `size(x,2)` retornam somente o número de linhas e colunas, respectivamente:

```
-->size(a,1)
 ans  =

    2.

-->size(a,2)
 ans  =

    5.
```

As funções `size(x,"r")` e `size(x,"c")` são, respectivamente, idênticas as funções anteriores:

```
-->size(a,"r")
 ans  =

    2.

-->size(a,"c")
 ans  =

    5.
```

Considere o vetor:

```
-->b = [3 1 2 4];
```

Apesar de ser um vetor, `b` é interpretado, a seguir, como uma matriz 1x4. Por exemplo:

```
-->size(b,"c")   //  devolve o numero de colunas de b
 ans  =

    4.
```

6.6 OPERAÇÕES ESCALAR-MATRIZ

Operações entre escalar e matriz seguem a regras comuns da matemática. Considere a matriz:

```
-->A = [1 2 3 4; 5 6 7 8]
 A  =

!   1.    2.    3.    4. !
!   5.    6.    7.    8. !
```

Exemplo de multiplicação:

```
-->2*A
 ans  =

!   2.     4.     6.     8.  !
!   10.    12.    14.    16. !
```

Exemplos de divisão:

```
-->A/4
 ans  =

!   0.25    0.5    0.75    1. !
!   1.25    1.5    1.75    2. !

-->A(:,2)/2
 ans  =

!   1. !
!   3. !
```

Exemplos de expressões:

```
-->A-2
 ans  =

! - 1.    0.    1.    2. !
!   3.    4.    5.    6. !

-->3*A-2
 ans  =

!   1.     4.     7.     10. !
!   13.    16.    19.    22. !

-->5+3*A(1,:)
 ans  =

!   8.    11.    14.    17. !
```

6.7 OPERAÇÕES MATRIZ-MATRIZ

Operações de adição e subtração entre matrizes seguem a regras comuns da matemática:

```
-->A = [1 2 3 4; 5 6 7 8]
 A  =

!   1.    2.    3.    4. !
!   5.    6.    7.    8. !

-->B = [3 1 3 8; 3 9 6 5]
 B  =

!   3.    1.    3.    8. !
!   3.    9.    6.    5. !

-->A+B
 ans  =

!   4.    3.     6.     12. !
!   8.    15.    13.    13. !

-->2*A-B
 ans  =

! - 1.    3.    3.    0.  !
!   7.    3.    8.    11. !
```

A operação de multiplicação também segue as regras comuns da matemática:

```
-->A = [1 2 3 4; 5 6 7 8]
 A  =

!   1.    2.    3.    4. !
!   5.    6.    7.    8. !

-->v = [2; 3; 4; 5]     // vetor-coluna
 v  =

!   2. !
!   3. !
!   4. !
!   5. !

-->A*v
 ans  =

!   40. !
!   96. !
```

Multiplicar matrizes de dimensões incompatíveis causam erros:

```
-->c = [2 3 5 1];

-->A*c
    !--error    10
inconsistent multiplication
```

Naturalmente, a multiplicação de `A` pela transposta de `c` é válida:

```
-->A*c'
 ans  =

!   27. !
!   71. !
```

O operador de **multiplicação pontuada** (.*) realiza uma multiplicação elemento por elemento entre matrizes. Considere as matriz:

```
-->A = [1 2 3 4; 5 6 7 8]
 A  =

!   1.    2.    3.    4. !
!   5.    6.    7.    8. !

-->B = [3 1 3 8; 3 9 6 5]
 B  =

!   3.    1.    3.    8. !
!   3.    9.    6.    5. !
```

Multiplicação pontuada:

```
-->A.*B
 ans  =

!   3.     2.     9.     32. !
!   15.    54.    42.    40. !
```

Do mesmo modo, a divisão elemento por elemento, requer o uso do ponto (divisão pontuada):

```
-->A./B
 ans  =

!   0.3333333    2.           1.           0.5 !
!   1.6666667    0.6666667    1.1666667    1.6 !
```

Existe também a potenciação elemento por elemento que também requer o uso do ponto (potenciação pontuada):

Em geral, o Scilab utiliza menos laços que as linguagens de programação tradicionais devido sua habilidade de substituir laços por alguma operação matricial.

```
-->[2 3 2 4].^[1 2 3 4]
 ans  =

    2.    9.    8.    256.
```

Exemplo de expressão:

```
-->A.^(2+1)+B/2
 ans  =

!   2.5      8.5      28.5    68.   !
!   126.5    220.5    346.    514.5 !
```

A seguinte expressão:

```
-->2.^A
 ans  =

!   2.     4.     8.      16.  !
!   32.    64.    128.    256. !
```

eleva 2 a cada um dos elementos de `A`.

Exercício resolvido 6.7.1. Calcular o valor da função $f(x) = \sin(x)\cos(x)$ para

$$x = \frac{\pi}{8}, \frac{2\pi}{8}, \frac{3\pi}{8}, \ldots, \frac{8\pi}{8}$$

Em muitas linguagens de programação este problema seria utilizar um laço FOR, como por exemplo:

```
1  x = %pi/8 : %pi/8 : %pi;
2  a = sin(x);
3  b = cos(x);
4  for i=1:8
5    y(i) = a(i)*b(i);
6  end
```

No Scilab é possivel substituir o laço FOR por uma multiplicação pontuada:

```
1  x = %pi/8 : %pi/8 : %pi;
2  y = sin(x).*cos(x);
```

6.8 SOLUÇÃO DE SISTEMAS DE EQUAÇÕES LINEARES

O Scilab foi especialmente projetado para simplificar cálculos de Algebra Linear. Um dos problemas mais comuns de Algebra Linear é a solução de sistemas de equações lineares:

$$\begin{cases} x_1 - x_2 + 2x_3 = 5 \\ x_1 - x_2 - 6x_3 = 0 \\ 4x_1 + x_3 = 5 \end{cases}$$

Este sistema também pode ser escrito na forma matricial $\mathbf{Ax} = \mathbf{b}$:

$$\begin{bmatrix} 1 & -1 & 2 \\ 1 & -1 & -6 \\ 4 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 5 \\ 0 \\ 5 \end{bmatrix}$$

Iniciamos a solução do sistema preparando as matrizes **A** e **b**:

```
-->A = [1 -1 2;  1 -1 -6; 4 0 1]
 A  =

!   1.  - 1.    2. !
!   1.  - 1.  - 6. !
!   4.    0.    1. !

-->b = [5 0 5]'            // A transposta é importante!
 b  =

!   5. !
!   0. !
!   5. !
```

O operador de **divisão à esquerda** \ resolve o sistema, isto é, o resultado de `A\b` é a solução do sistema:[1]

```
-->A\b
 ans  =

!   1.09375 !
! - 2.65625 !
!   0.625   !
```

6.9 TRANSPOSTA DE MATRIZES COMPLEXAS

Se a matriz possue números complexos, o operador de transposição (') produz a transposta conjugada, ou seja, aplica a transposição e a conjugação complexa na matriz. Por exemplo:

```
-->a = [%i 2+%i 5; -1+2*%i 3*%i %i]
 a  =

!   i           2. + i      5.  !
! - 1. + 2.i    3.i         i   !

-->a'
 ans  =
```

[1] O operador de **divisão a direita** / é definido em termos do operador de divisão à esquerda: `A/b = (A'\b')'`.

Tabela 6.3: Transposta de Matrizes Complexas

Seja `A` e `B` matrizes reais.
Considere a matriz complexa:
`Z = A + %i*B`
Então:

```
Z'  = A' - %i*B'
Z.' = A' + %i*B'
```

```
! - i         - 1. - 2.i !
!   2. - i    - 3.i      !
!   5.        - i        !
```

Use o operador de transposição pontuada (.') para produzir a transposta sem operação de conjugação:

```
-->a.'
 ans  =

!   i         - 1. + 2.i !
!   2. + i      3.i      !
!   5.          i        !
```

Um resumo é mostrado na Tabela 6.3

6.10 ZEROS E ONES

Capítulo 7

FUNÇÕES

7.1 INTRODUÇÃO

Quando o tamanho de um programa estende-se a centenas de linhas, o programa torna-se difícil de compreender e administrar. Por isso, dividir um grande programa computacional em partes menores para facilitar a compreensão (ou legibilidade) do problema é uma tarefa comum em programação de computadores. No Scilab, este trecho menor do programa é chamado de função. Funções são também chamadas de sub-rotinas, módulos, subprogramas ou subalgoritmos.

Funções são usados também para evitar repetição do mesmo código no programa. Por exemplo, suponha que seu programa tenha a tarefa de por em ordem crescente várias listas de números. Em vez de repetir o código toda vez que for realizar esta tarefa, você escreve uma função para ordenar listas numéricas e depois chama a mesma função sempre que for ordenar uma lista. Neste sentido, as funções apresentam as seguintes vantagens: a) Você escreve o código somente uma vez. b) Você pode reutilizar a função em outros programas. c) Uma vez que você tem corrigido todos os erros do programas (i.e., depurado o programa), ele funcionara corretamente não importa quantas vezes você use a função.

Em resumo, funções são usadas para:

1. Dividir um grande programa em programas menores;
2. Repetir uma tarefa que é realizada frequentemente sem ter que repetir o mesmo código em vários lugares;
3. Aumentar a legibilidade do programa.

No Scilab já existem muitas funções prontas (**pré-definidas**), algumas delas elementares (`cos(x)` e `sin(x)`) e outras específicas para aplicações em engenharia, matemática, física, e na estatística. O objeto de estudo deste capítulo são **as funções definidas pelo usuário**. Isto é, funções que são elaboradas pelos próprios usuários do Scilab.

7.2 PARÂMETROS DE ENTRADA E SAÍDA

As funções recebem dados por meio de uma lista de **parâmetros de entrada**, e retorna resultados por uma lista de **parâmetros de saída**. Por exemplo, a função `cos(x)` recebe um valor e retorna um valor, logo tem um parâmetro de entrada e um de saída. A função, `modulo(x,y)` recebe dois valores (o numerador e o denominador) e retorna um valor (o resto). Logo, `modulo(x,y)` têm dois parâmetros de entrada e um de saída. A função size(x), tem um parâmetro de entrada e dois de saída. Por exemplo,

```
-->a = [1 2 3; 4 5 6]
 a  =

!   1.    2.    3. !
!   4.    5.    6. !

-->[l c] = size(a)
 c  =

    3.
 l  =

    2.
```

porque recebe uma matriz e devolve dois valores (o número de linhas e colunas).

7.3 FUNÇÕES DEFINIDAS PELO USUÁRIO

A forma geral de uma função é:

```
function [y1,y2,...,ym] = nomedafuncao(x1,x2,x3,...,xn)

<comandos>...

endfunction
```

Onde,

`function`	Palavra reservada que indica o inicio de uma função.
`nomedafuncao`	o nome da função é definido pelo usuário.
`x1, x2, x3,..., xn`	parâmetros de entrada.
`y1, y2, y3,..., ym`	parâmetros de saída.
`<comandos>`	Comandos do Scilab a serem executados pela função.

A declaração:

```
function [y1,y2,...,ym] = nomedafuncao(x1,x2,x3,...,xn)
```

é o **cabeçalho da função** e serve, entre outras coisas, para dar o nome da função e definir a lista de parâmetros de entrada e saída (também chamados de **parâmetros formais**).

Quando há apenas um parâmetro de saída, os colchetes podem ser omitidos. Por exemplo, a seguinte função tem apenas um parâmetro de saída e um parâmetro de entrada. Esta função calcula o fatorial de um número:

```
1 function y = fat(n)
2   p = 1;
3   for i = n:-1:2
4     p = p*i;
5   end
6   y = p;
7 endfunction
```

Na linha de comando `y = p` da função `fat(x)` é atribuído o valor a ser retornado pelo parâmetro de saída `y`.

☞ **IMPORTANTE**: Cada parâmetro da lista de parâmetros de saída de uma função é necessário aparecer a esquerda de pelo menos um comando de atribuição da função.

Quando a função termina, o valor contido nos parâmetros de saída são retornados ao programa que chamou a função.

Uma **chamada** (ou **ativação**) de função é a solicitação explícita para executar uma função. A seguinte chamada de função no console do Scilab executa a função `fat`:

```
-->fat(4)
 ans  =

    120.
```

Quando ocorre a chamada `fat(5)`, o valor 5 é passado para a variável `n` (o parâmetro de entrada de `fat`). Deste modo, a função `fat` calcula o fatorial de `n` (igual a 5) e retorna o valor contido no parâmetro de saída `y`, ou seja, 120.

É comum criarmos functions em um arquivo de script separado para ser utilizado em vários programas. Para isto digitamos a função em um arquivo com extensão .SCI. Por exemplo, fatorial.sci. Em seguida, usamos a função `exec()` do Scilab para carregar o arquivo fatorial.sci para dentro do Scilab. Por exemplo:

```
--> exec("fatorial.sci");
```

Neste comando, foi assumido que o arquivo fatorial.sci foi salvo no diretório atual do Scilab. Use a opção **ARQUIVO** » **ALTERAR O DIRETÓRIO ATUAL** ou o comando `chdir()` para mudar o diretório atual do Scilab.

Uma função definida pelo usuário tem o mesmo status de uma função pré-definida do Scilab e, portanto, pode ser usada do mesmo modo. Vejamos um exemplo.

Exercício resolvido 7.3.1. Calcular o seguinte somatório usando a função fat definida acima.

$$S = \sum_{i=1}^{10} i! = 1! + 2! + \ldots + 10!$$

Solução:

```
1  soma = 0;
2  for j=1:10
3    soma = soma + fat(j);
4  end
```

Comentário. Para executar este programa, o usuário deve, antes, carregar a função `fat()`. Isto pode ser feito através, por exemplo, da função `exec()`.

7.4 A IDÉIA BÁSICA DAS FUNÇÕES

Com as funções é possível escrever um programa e chama-lo quantas vezes quiser em diferentes pontos de um outro programa, geralmente, usando diferentes parâmetros de entrada. Se uma função é chamada, o controle de execução dos comandos é transferido para a função. Quando a função termina, o controle é devolvido ao programa chamador *no mesmo local* em que a função foi originalmente chamada (ver Figura 7.1). E o programa chamador continua executando os seus comandos a partir da linha imediatamente depois da chamada da função.

Considere as seguintes funções

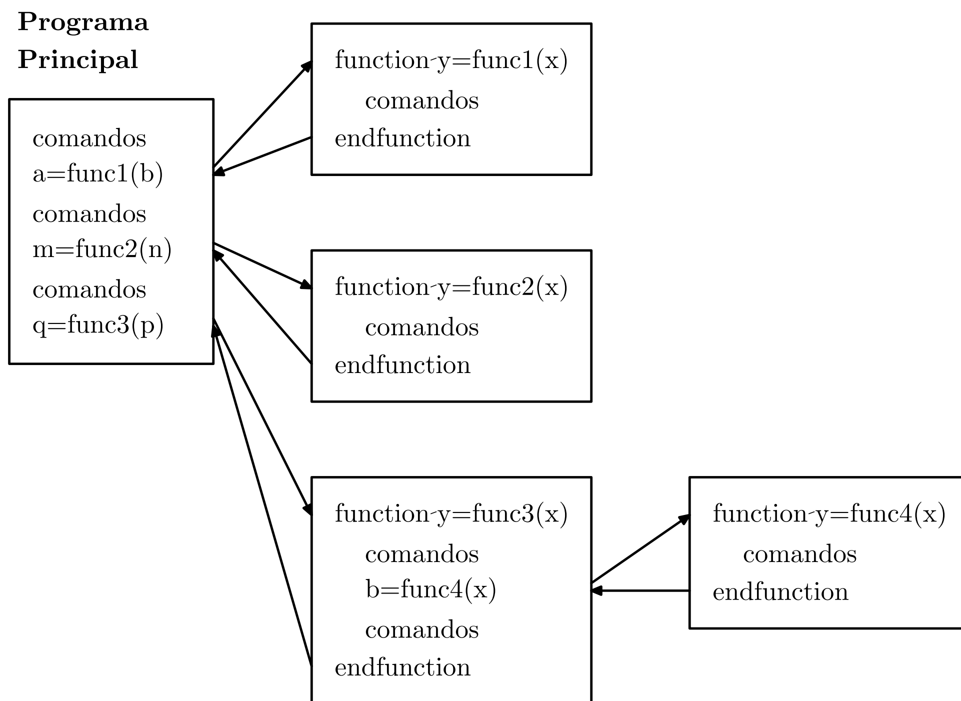


Figura 7.1: O programa principal chama as funções func1(), func2() e func3(). O controle é transferido para as funções, mas sempre retorna ao programa principal. A função func3() chama func4() transferindo o controle para func4(). A função func4(), quando termina, retorna o controle ao programa chamador (que é func3()).

```
1  function y = soma(x)
2    n = length(x);   // calcula o tamanho do vetor
3    s = 0;
4    for i=1:n
5      s = s + x(i);
6    end
7    y = s;
8  endfunction
```

```
1  function y = media(x)
2    n = length(x);
3    y = soma(x)/n;  // chama a função soma
4  endfunction
```

E o programa principal que chama as funções.

```
1  ne = input("Qual o numero de elementos do vetor");
2  for i=1:ne
3    a(i) = input("digite um elemento");
4  end
5  s = soma(a);
6  m = media(a);
7  printf("soma = %g\n",s);
8  printf("media = %g\n",m);
```

No programa principal, a chamada da função `soma()` (na linha 5) transfere o fluxo de controle para a função que, ao terminar, devolve o controle para o programa chamador.

O programa principal executa o comando da linha seguinte (linha 6) e chama a função `media()`. A função `media()` chama a função `soma()` (linha 3 da função `media()`) que devolve o controle para a função `media()` que então devolve o controle para o programa chamador.

7.5 ESCOPO DE VARIÁVEIS

Relembremos que as variáveis definidas no console do Scilab e pelos scripts são armazenadas em uma área da memória chamada de espaço de trabalho. Por exemplo,

```
--> clear;           // apaga todas as variáveis do espaço de trabalho

--> x = 2;

--> y = 3;
```

Como `x` e `y` estão na memória, estão o comando

```
--> z = x + y;
```

armazena 5 em `z`. Porém, seguinte comando causará um erro:

```
--> a = x + y + w
```

por que a variável `w` não está no espaço de trabalho (uma vez não foi definida no console). As variáveis definidas pelas funções também não são armazenadas no espaço de trabalho. Deste modo, estas variáveis não são visíveis no console. Por exemplo, a variável `p` da função `fat()` não pode ser usada no console:

```
--> fat(3)

--> p + 1           // gera erro
```

porque `p` não esta definida no espaço de trabalho. Neste caso, dizemos que `p` é uma **variável local** da função `fat()`. A seguir introduziremos os conceitos de variáveis locais e o globais.

7.5.1 Variáveis Locais

Uma variável é dita ser *local* quando é definida dentro de uma função. Toda *variável local* deixa de existir (torna-se inválida) quando a função é finalizada. Por isso, a variável local é dita ser visível localmente na função. As variáveis locais também não podem alterar as variáveis do espaço de trabalho.

Vejamos um exemplo. Considere a seguinte função:

```
1  function y = beta(x)
2      a = 3;
3      b = 2;
4      c = 5;
5      printf("a = %g  b = %g  c = %g\n",a,b,c);
6      y = a + b + c + x;
7  endfunction
```

e programa principal que usa esta função

```
a = 23;
b = 50;
c = 200;
w = beta(2);
printf("a = %g  b = %g  c = %g\n",a,b,c);
```

Quando nós executamos o programa principal no Scilab obtemos a seguinte resposta:

```
a = 3  b = 2  c = 5
a = 23  b = 50  c = 200
```

Note que dentro da função beta as variáveis a, b e c possuem os valores 3, 2, e 5 respectivamente. Ao passo que, fora da função beta, as variáveis a, b e c possuem os valores 23, 50, e 200 respectivamente. A explicação desta discordância é a seguinte: as variáveis a, b, e c dentro da função beta são locais e por isso deixam de existir tão logo a função termine. Scilab armazena as variáveis locais da função em uma local diferente das do espaço de trabalho. Portanto, as variáveis do espaço de trabalho (a = 23, b = 50, c = 200) não são alteradas pela função.

O fato de que as variáveis locais só poderem ser usadas internamente pela função elimina qualquer conflito que possa surgir, caso um programa (ou outras funções) resolva utilizar os mesmos nomes de variáveis da função. Uma forma de evitar este comportamento é usar as variáveis globais.

7.5.2 Variáveis Globais

Para alterar as variáveis do programa principal, temos que transforma-las em variáveis globais usando a declaração `global`. Por exemplo, considere a seguinte função:

```
function y = gama(x)
    global R;
    global S;
    R = 1;
    S = 2;
    t = 3;
    printf("R = %g  S = %g  t = %g\n",R,S,t);
    y = R + S + t + x;
endfunction
```

e programa principal que usa esta função

```
global R;
global S;
R = 5;
S = 10;
t = 15;
u = gama(2);
printf("R = %g  S = %g  t = %g\n",R,S,t);
```

Quando nós executamos o programa principal no Scilab obtemos a seguinte resposta:

```
R = 1  S = 2  t = 3
R = 1  S = 2  t = 15
```

Note que as variáveis `R` e `S` do programa principal foram alteradas dentro da função `gama()`. Isto aconteceu porque usamos a comando `global`. Porém a variável `t` não foi alterada pela função `gama()` porque não é uma variável global, ou seja, a variável `t` não foi declarada como global, permanecendo como uma variável local.

Concluímos que variáveis locais são visíveis somente dentro na função, mas variáveis globais podem ser visíveis tanto dentro como fora de uma função. Variáveis globais são, portanto, uma forma de compartilhar uma variável entre a função e o programa chamador.

A declaração `global` deve ser usado tanto no programa chamador como na função (e em qualquer função que venha a compartilhar a mesma variável). A declaração `global` é opcional quando uma variável, apesar de ser global, não é modificada pela função. Por exemplo, considere a função:

Boa programação: use caixa alta para nomes de variáveis globais para tornar claro ao leitor que são globais e para não confundir com variáveis locais.

```
1  function y = eta(x)
2      global S
3      S = x+3;
4      y = R + S;
5  endfunction
```

e o programa principal que usa esta função:

```
1  global S
2  R = 5;
3  S = 10;
4  u = eta(2);
5  printf("u = %g \n",u);
```

Claramente, as variáveis `R` e `S` são globais, mas somente a variável `S` foi declarada `global` por que ela é modificada pela função `eta()`. A variável global `R` não precisa ser declarada `global` porque não é modificada pela função `eta()`.

7.6 OS PROGRAMAS DO SCILAB

Existem no Scilab dois tipos de programas:

1. Arquivo de comandos.
   (a) É uma sequência de comandos do Scilab;
   (b) Possuem extensão de arquivo .SCE (mas não é obrigatório);
   (c) É executado pelo comando exec();
   (d) Armazena suas variáveis em uma área chamada Espaço de Trabalho (Workplace).
2. Arquivo de funções.
   (a) É uma sequência de functions (sub-rotinas);
   (b) Possuem extensão de arquivo .SCI;
   (c) É carregado pelo comando exec();
   (d) Armazena suas variáveis em uma área própria. Essas variáveis são chamadas de locais.

7.7 PASSAGEM DE PARÂMETROS

Funções podem ter zero, um ou mais de um parâmetros de entrada. Por exemplo, uma função com o seguinte cabeçalho possui um parâmetro de entrada:

```
function x = fatorial(n)
```

Esta outra function tem três parâmetros de entrada:

```
function x = zeta(a,b,c)
```

Esta função não tem parâmetros de entrada:

```
function y = psi()
```

Um parâmetro de saída pode ser um número, um vetor ou uma matriz. A seguinte Função possui três parâmetros de saída (e três de entrada):

```
1  function [x, y, z] = beta(a, b, c)
2    a = a/2;
3    b = b/2;
4    c = c/2;
5    printf("a=%g b=%g c=%g\n",a,b,c);
6    x = a;
7    y = b;
8    z = c;
9  endfunction
```

Considere o seguinte programa principal

```
1    a = 10;
2    b = 20;
3    c = 30;
4    r1 = 2;
5    r2 = 4;
6    r3 = 6;
7    [d1, d2, d3] = beta(r1,r2,r3)
8    printf("a=%g b=%g c=%g\n",a,b,c);
9    printf("d1=%g d2=%g d3=%g\n",d1,d2,d3);
```

Este programa chama `beta()` e passa os parâmetros `r1`, `r2` e `r3` do seguinte modo: o valor de `r1` é colocado em `a`, o valor de `r2` é colocado em `b` e o valor de `r3` é colocado em `c`. Do mesmo modo, a função retorna os parâmetros `x`, `y` e `z` para as variáveis `d1`, `d2` e `d3`, respectivamente. O resultado do programa é

```
Resultado
a=1 b=2 c=3
a=10 b=20 c=30
d1=1 d2=2 d3=3
```

7.8 EXEMPLOS

Exercício resolvido 7.8.1. Ler três pontos (x_1, y_1), (x_2, y_2) e (x_3, y_3) do plano cartesiano representando os vértices de um triângulo. Calcular a área do triângulo.

Da geometria, tem-se o seguinte fato: se a, b e c são as medidas dos lados de um triângulo, então a área deste triângulo é dada por:

$$\text{area} = \sqrt{s(s-a)(s-b)(s-c)}$$

onde s é o semi-perímetro dado por:

$$s = \frac{a+b+c}{2}$$

Para calcular os lados do triangulo será usado a função `dist` para calcular a distância entre dois pontos (x_1, y_1) e (x_2, y_2) no plano cartesiano:

```
1  function d = dist(x1,y1,x2,y2)
2    d = sqrt((x2-x1)^2+(y2-y1)^2);
3  endfunction
```

Usando as fórmulas acima, uma solução para o problema é a seguinte:

```
1  x1 = input("digite x1");
2  y1 = input("digite y1");
3  x2 = input("digite x2");
4  y2 = input("digite y2");
5  x3 = input("digite x3");
6  y3 = input("digite y3");
7
8  // Cálculo dos lados do triângulo
9
10 a = dist(x1,y1,x2,y2);   // medida do lado A
11 b = dist(x1,y1,x3,y3);   // medida do lado B
12 c = dist(x2,y2,x3,y3);   // medida do lado C
13
14 s = (a+b+c)/2;           // semiperimetro
15
16 area = sqrt(s*(s-a)*(s-b)*(s-c));
17
18 printf("Área do triângulo = %g\n",area);
```

Exercício resolvido 7.8.2. Elaborar uma função `inverte()` que receba um vetor X. A função retorna um vetor x invertido. Por exemplo, se a função recebe |[2 1 8 5]|, ela retorna `[5 8 1 2]`.

Solução:

```
1  function y = inverte(x)
2    n = length(x);
3    for i=1:n
4      y(i) = x(n+1-i);
5    end
6  endfunction
```

Exercício resolvido 7.8.3. Muitas funções matemáticas podem ser calculadas por meio de um somatório infinito de termos. Em cada caso, a precisão aumenta à medida que mais termos na série são considerados. Um exemplo, é a função $\cos x$:

$$\cos x = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} +$$

Para cálculos práticos, este somatório infinito devem terminar após um número finito de termos (penalizando a precisão do resultado). Preparar uma função para calcular o co-seno (função COSENO(x, n)), com duas variáveis de entrada, onde a primeira variável de entrada é x e a segunda variável de entrada é o número de termos a serem utilizados nos cálculos.

Solução

```
function y = coseno(x,n)
   s = 1;
   for i=1:n
     num = x^(2*i);
     den = fat(2*i);
     sinal = (-1)^i;
     s = s + sinal*num/den;
   end
   y = s;
endfunction
```

Função auxiliar para calcular o fatorial:

```
function x = fat(n)
   p = 1;
   for i=n:-1:2
      p = p*i;
   end
   x = p;
endfunction
```

Exercício resolvido 7.8.4. Elaborar uma função `membro` que receba um número e um vetor. Uma função retorna o valor `%t` se o número existe no vetor. Caso contrário, a função retorna `%f`.

7.9 O COMANDO RETURN

Normalmente uma função termina após executar a última linha. O comando return, porém, pode interromper a execução de uma function em qualquer ponto do programa. Por exemplo,

```
function [maxdc, indic] = mdc(a, b)
// Esta função calcula o máximo divisor de dois
// números a e b positivos.
// indic retorna 1 se o cálculo do m.d.c. teve êxito
//       retorna 0 se os dados de entrada foram
//       inadequados.

indic = 0;
maxdc = 0;
if round(a) <> a | round(b) <> b
    return;    // Aqui o comando return interrompe
               // o programa.
end
if a < 1 | b < 1
    return;    // Aqui também o comando return
               // interrompe o programa.
end
if a < b
    t = a;
    a = b;
    b = t;
end
indic = 1;
r = 1;
```

```
25 while r <> 0
26    r = modulo(a, b);
27    a = b;
28    b = r;
29 end
30 maxdc = a;
31 endfunction
```

7.10 ESTUDO DE CASO: UM PROGRAMA DE ESTATÍSTICA

Será elaborado nesta seção um programa para calcular as seguintes estatísticas de conjuntos de valores digitados pelos usuário e armazenado no vetor $x = (x_1, x_2, \ldots, x_n)$.

$$
\begin{aligned}
\text{soma}(x) &= \sum_{i=1}^{n} x_i \\
\text{média}(x) &= \frac{1}{n}\sum_{i=1}^{n} x_i \\
\text{variância}(x) &= \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \text{média}(x))^2 \\
\text{desvio padrão}(x) &= \sqrt{\text{variância}(x)}
\end{aligned}
$$

O primeiro programa de estatística mostrado abaixo não emprega a técnica da sub-rotina. Tornando-se grande e mais complexo.

PROGRAMA VERSÃO 1 - Sem sub-rotinas

```
1  printf("Meu Programa de Estatística Versao 1");
2  n = input("Digite o numero de elementos");
3  for i=1:n
4     x(i) = input("Digite um número entre 1 e 10");
5  end
6  printf("Opcao 1 - Soma\n");
7  printf("Opcao 2 - Média\n");
8  printf("Opcao 3 - Desvio padrão\n");
9  printf("Opcao 4 - Variância\n");
10 opcao = input("Digite sua opcao");
11
12 if opcao == 1
13    soma = 0;
14    for i=1:n
15       soma = soma + x(i);
16    end
17    printf("A soma é igual a %g",soma);
18 end
19
20 if opcao == 2
21    soma = 0;
22    for i=1:n
23       soma = soma + x(i);
24    end
25    media = soma/n;
26    printf("A média é igual a %g",media);
```

```
27 end
28
29 if opcao == 3
30    soma = 0;
31    for i=1:n
32       soma = soma + x(i);
33    end
34    media = soma/n;
35    d = 0;
36    for i = 1:n
37       d = d + (x(i)-media)^2;
38    end
39    dpad = sqrt(d/(n-1));
40    printf("O desvio padrão é igual a %g",dpad);
41 end
42
43 if opcao == 4
44    soma = 0;
45    for i=1:n
46       soma = soma + x(i);
47    end
48    media = soma/n;
49    d = 0;
50    for i = 1:n
51       d = d + (x(i)-media)^2;
52    end
53    var = d/(n-1);
54    printf("A variância é igual a %g",var);
55 end
```

O programa 2, mostrado a seguir, emprega a técnica de modularização, que divide o programa em partes menores tornando-o mais fácil de compreender.

PROGRAMA VERSÃO 2 - Com sub-rotinas

O programa principal:

```
1  printf("Meu Programa de Estatística Versao 2");
2
3  n = input("Digite o número de elementos");
4  for i=1:n
5      x(i) = input("Digite um número entre 1 e 10");
6  end
7
8  printf("Opção 1 - Soma\n");
9  printf("Opção 2 - Média\n");
10 printf("Opção 3 - Desvio padrão\n");
11 printf("Opção 4 - Variância\n");
12 opcao = input("Digite sua opção");
13
14 if opcao == 1
15     printf("A soma é igual a %g",soma(x));
16 end
17
18 if opcao == 2
19     printf("A média é igual a %g",media(x));
20 end
21
```

```
22  if opcao == 3
23      printf("O desvio padrão é igual a %g",dpad(x));
24  end
25
26  if opcao == 4
27      printf("A variância é igual a %g",var(x));
28  end
```

As sub-rotinas:

```
1   // Função soma
2   function y = soma(x)
3   n = length(x);
4   s = 0;
5   for i=1:n
6       s = s + x(i);
7   end
8   y = s;
9   endfunction
10
11  // Função média
12  function y = media(x)
13  n = length(x);
14  y = soma(x)/n;
15  endfunction
16
17  // Função variância
18  function y = var(x)
19  n = length(x);
20  m = media(x);
21  d = 0;
22  for i = 1:n
23      d = d + (x(i)-m)^2;
24  end
25  y = d/(n-1);
26  endfunction
27
28  // Função Desvio Padrão
29  function y = dpad(x)
30  y = sqrt(var(x));
31  endfunction
```

Note que para calcular a variância foi usado a função média, evitando assim a repetição de código. Aliás, como já foi dito, evitar a repetição de código é uma das vantagens de usar sub-rotinas. A função `length(x)` usada no código acima é explicada na próxima seção.

7.10.1 O Comando de Múltipla Escolha SELECT-CASE

O comando SELECT-CASE é conveniente para testar se uma expressão é igual a uma lista de valores diferentes. A sintaxe do SELECT-CASE é:

```
select <expressão>
  case <valor1> then
     <comandos> ...
  case <valor2> then
     <comandos> ...
```

```
  case <valor i> then
     <comandos> ...
  else
     <comandos> ...
  end
end
```

Se <expressão> for igual a <valor1> então o primeiro case é executado. Se <expressão> for igual a <valor2> então o segundo case é executado. E assim por diante. O comando else é executado se todas as comparações do comando case forem falsas.

PROGRAMA VERSÃO 3 - Com o comando SELECT-CASE

```
1  printf("Meu Programa de Estatística Versao 3");
2
3  n = input("Digite o número de elementos");
4  for i=1:n
5      x(i) = input("Digite um número entre 1 e 10");
6  end
7
8  printf("Opção 1 - Soma\n");
9  printf("Opção 2 - Média\n");
10 printf("Opção 3 - Desvio padrão\n");
11 printf("Opção 4 - Variância\n");
12 opcao = input("Digite sua opção");
13
14 select opcao
15 case 1 then
16     printf("A soma é igual a %g\n",soma(x));
17 case 2 then
18     printf("A média é igual a %g\n",media(x));
19 case 3 then
20     printf("O desvio padrão é igual a %g\n",dpad(x));
21 case 4 then
22     printf("A variância é igual a %g\n",var(x));
23 end
```

A seguir o programa 3 é incrementado em dois aspectos:

1. Usa um loop que repete o menu até o usuário digitar a opção 0;
2. Verifica se o usuário digitou uma opção inválida usando o comando ELSE.

PROGRAMA VERSÃO 4 - Versão Final

```
1  printf("Meu Programa de Estatística Versao 4");
2  n = input("Digite o número de elementos");
3  for i=1:n
4      x(i) = input("Digite um número entre 1 e 10");
5  end
6
7  opcao = 1;
8  while opcao <> 0
9      printf("\n");
10     printf("Opção 1 - Soma\n");
11     printf("Opção 2 - Média\n");
12     printf("Opção 3 - Desvio padrão\n");
13     printf("Opção 4 - Variância\n");
```

```
14        printf("Opção 0 - Fim\n");
15        opcao = input("Digite sua opção");
16        select opcao
17        case 1
18            printf("A soma é igual a %g\n",soma(x));
19
20        case 2
21            printf("A média é igual a %g\n",media(x));
22
23        case 3
24            printf("O desvio padrão é igual a %g\n",dpad(x));
25
26        case 4
27            printf("A variância é igual a %g\n",var(x));
28
29        case 0
30            printf("Até a logo\n");
31
32        else
33            printf("Você digitou uma opção inválida\n");
34        end
35    end
```