

A linguagem Fortran

- Em 1954, a linguagem de alto nível Fortran foi proposta por um grupo da IBM.
- O primeiro **compilador** (ou seja, um programa que traduz programas escritos em linguagem de alto nível para instruções de máquina) foi naturalmente escrito em Assembler.
- A máquina era um IBM 704: um computador com 15K de memória.

1

Linguagens de programação

- Existem várias linguagens de programação que descendem do Fortran; por exemplo:
 - 1959 – Cobol;
 - 1964 – Basic;
 - 1970 – Pascal;
 - 1971 – C;
 - 1983 – C++;
 - 1991 – Python;
 - 1995 – Java;
 - 1995 – PHP.

2

Matlab

- Foi criado no fim dos anos 70 por *Cleve Moler* e lançado comercialmente em 1984 pela empresa *MathWorks*.
- É voltado para engenheiros e cientistas.
- Possui grande facilidade para o tratamento de matrizes (MatLab = *Matrix Laboratory*).
- É um **interpretador**, ou seja, um programa que executa programas; ao contrário de um *compilador*, não traduz um programa para instruções de máquina.

3

Scilab

- Foi criado em 1990 por pesquisadores do INRIA e da École Nationale des Ponts et Chaussées (França), sendo gratuito e bastante semelhante ao MatLab.
 - <http://www.scilab.org>
- Consiste também em um interpretador.
- A linguagem e o sistema possuem o mesmo nome: Scilab.
- Será apresentada a versão 5.1 do Scilab.

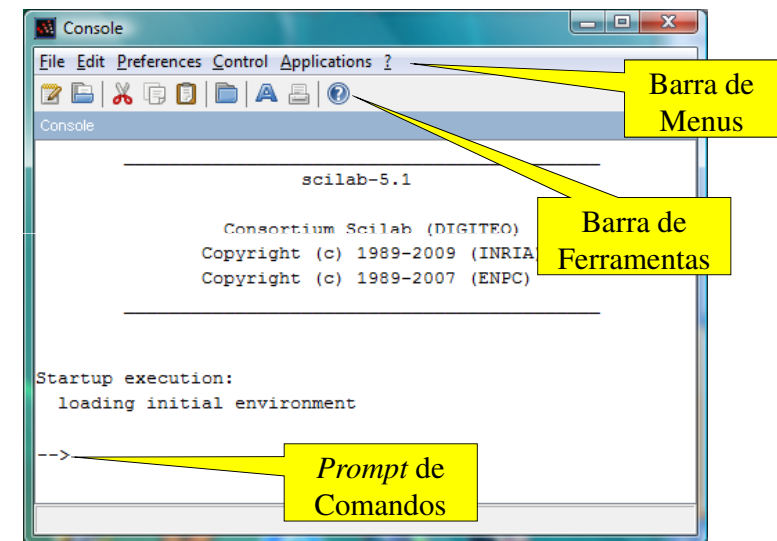
4

O ambiente e a linguagem Scilab

- O ambiente Scilab:
 - interpreta comandos;
 - oferece um editor para a construção de programas (SciPad);
 - emite mensagens de erros relativos à obediência da sintaxe da linguagem e a problemas na execução de um programa (como divisão por zero).
- Como qualquer linguagem natural, a linguagem Scilab:
 - une riqueza de expressão a detalhes sintáticos;
 - exige uma postura paciente em seu aprendizado, pois envolve uma taxa inicial de memorização; a fluência vem com a prática.

5

O ambiente Scilab



6

Variáveis e comandos de atribuição

Prompt de Comandos points to the '-->' prompt.

```

-->a = 10
a =
10.
-->b = 2^10
b =
1024.
-->c = a+b
c =
1034.
  
```

"a" é uma *variável* que passa a existir; no caso, é atribuído o valor 10 à mesma points to the variable 'a'.

O Scilab mostra o valor recebido pela variável points to the value '10.'.

Exponenciação points to the expression '2^10'.

O valor atribuído pode ser uma expressão aritmética com variáveis já conhecidas points to the expression 'a+b'.

7

Variáveis

- Variáveis correspondem a nomes para espaços de memória que são gerenciados pelo Scilab.
 - O programador não precisa ter qualquer idéia de como tal gerência é realizada.
- Os nomes das variáveis são escolhidos pelo programador, respeitando as seguintes regras:
 - o primeiro caractere do nome deve ser uma letra ou qualquer caractere dentre '%', '_', '#', '!', '\$' e '?';
 - os outros caracteres podem ser letras ou dígitos ou qualquer caractere dentre '_', '#', '!', '\$' e '?'.

8

Variáveis

■ Nomes válidos:

- `a`, `A`, `jose`, `total_de_alunos`, `#funcionarios`.

■ Nomes inválidos:

- `1Aluno` (o primeiro caractere é um algarismo);
- `total de alunos` (tem espaços);
- `José` (é acentuado).

Comando de atribuição

■ Sintaxe:

<variável> = <expressão>

- A <variável>, se não existia, passa a existir.
 - Se existia, o valor armazenado anteriormente é perdido.
- A <expressão> é calculada e o resultado é atribuído à <variável>.

Variáveis e comandos de atribuição

```
-->d = a+x
!--error 4
Undefined variable: x
```

```
-->b = 2*b
b =
2048.
```

A expressão pode conter a própria variável

“*” denota multiplicação

Todas as variáveis em uma expressão devem ser definidas anteriormente

Variáveis e comandos de atribuição

```
-->a = %pi
a =
3.1415927
```

Constante aritmética: valor pré-definido de π

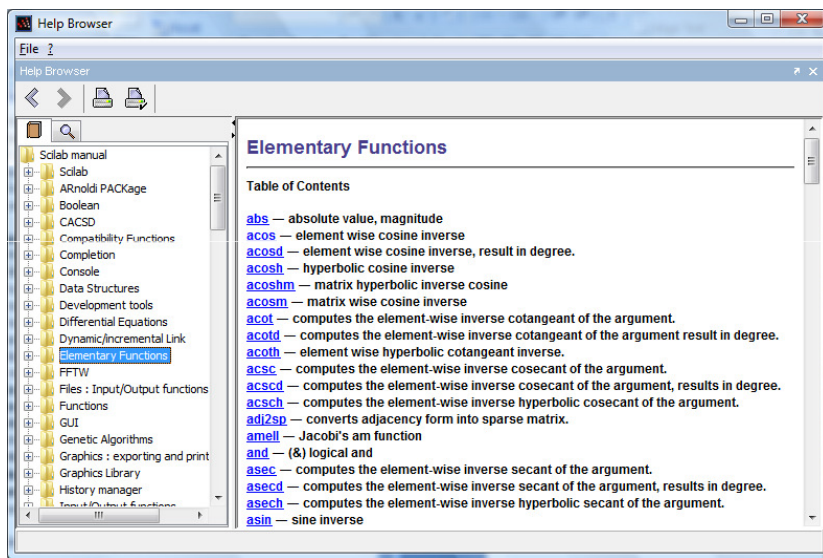
```
-->b = 2*pi
b =
6.283185307
```

“;” suprime a apresentação automática do valor

```
-->c = cos(a) + sqrt(b)
c =
1.5066283
```

O Scilab oferece um número de funções pré-definidas

Help - Funções elementares do Scilab



13

Expressões aritméticas

- Expressões podem ser arbitrariamente complicadas.
 - Qual é o valor de "x" a partir do comando " $x = 2^3 * 4$ " ?
 $2^3 \times 4 = 32$ ou $2^{3 \times 4} = 4096$?

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

- Parênteses podem alterar prioridades.

14

Prioridades e parênteses

```

-->2^3*4
ans = 32.
-->2^(3*4)
ans = 4096.
-->2^3^4
ans = 2.418D+24
-->2^(3^4)
ans = 2.418D+24
-->(2^3)^4
ans = 4096.
-->2*3+4
ans = 10.
-->2*(3+4)
ans = 14.

```

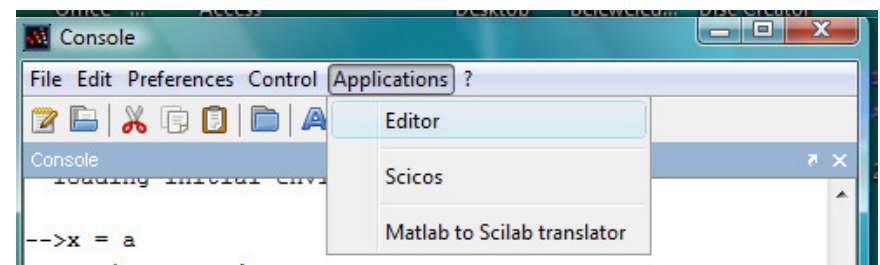
Recomendação:
use parênteses por
ser mais seguro.

Notação Scilab (e Fortran,
e C, e Java, e ...) para
 2.418×10^{24}

15

Programas Scilab

- Programas em Scilab são arquivos ASCII (caracteres sem formatação) com a terminação ".sce".
 - Um arquivo-programa contém comandos Scilab.
- Um programa é construído usando o editor SciPad.



16

Programas Scilab

- Use sempre o SciPad para construir programas
 - Nunca use o *Word*, pois ele introduz "caracteres" de formatação.
- Um programa é executado seguindo o menu "Execute/Load into Scilab" do editor SciPad.
 - Sua execução equivale à digitação na console dos comandos do programa.

17

Programa: equação de segundo grau – 1ª versão

```

SciPad 7.18.1 - Eq2g1.sce
File Edit Search Execute Scheme Options
Windows Help

1 // Cálculo das raízes de uma equação
2 // de 2o grau
3
4 a = 1
5 b = 2
6 c = 3
7 delta = b^2 - 4*a*c
8 r1 = (-b+sqrt(delta))/(2*a)
9 r2 = (-b-sqrt(delta))/(2*a)
10
Line: 9 Column: 28 Logical line: 9
  
```

“//” indica que a linha é um comentário

18

Programa: equação de segundo grau – 2ª versão

- Para uma nova equação, basta substituir no programa os valores dos coeficientes.
- Entretanto, a prática de modificar programas, a cada execução, não é recomendada.
 - O melhor é fazer com que o programa *leia* os valores dos coeficientes a cada execução.

```

// Entrada dos coeficientes
a = input("Entre com o valor de a:");
b = input("Entre com o valor de b:");
c = input("Entre com o valor de c:");
  
```

Diálogo com o usuário

19

Programa: equação de segundo grau – 2ª versão

```

// Cálculo das raízes de uma equação
// de 2o grau

a = input("Entre com o valor de a:")
b = input("Entre com o valor de b:")
c = input("Entre com o valor de c:")

delta = b^2 - 4*a*c
x1 = (-b+sqrt(delta))/(2*a)
x2 = (-b-sqrt(delta))/(2*a)
  
```

20

Execução do programa anterior

```

Entre com o valor de a:1
a =
    1.
Entre com o valor de b:2
b =
    2.
Entre com o valor de c:3
c =
    3.
delta =
    - 8.
x1 =
    - 1. + 1.4142136i
x2 =
    - 1. - 1.4142136i
  
```

21

Programa: equação de segundo grau – 3ª versão

■ Especificação:

- Para que a equação seja do segundo grau, o coeficiente "a" deve ser diferente de 0.
- De acordo com o valor de delta, o resultado do programa deve ser:
 - uma mensagem informando que as raízes não são reais ($\Delta < 0$);
 - uma única raiz real ($\Delta = 0$);
 - duas raízes reais ($\Delta > 0$).

22

Comando condicional if

```

if <condição> then
    <bloco "então">
else
    <bloco "senão">
end
  
```

```

if <condição> then
    <bloco "então">
end
  
```

23

Programa: equação de segundo grau – 3ª versão

```

// Cálculo das raízes de uma equação
// de 2o grau

a = input("Entre com o valor de a:");
if (a == 0) then
    printf("O coeficiente a deve ser diferente de 0.\n");
else
    b = input("Entre com o valor de b:");
    c = input("Entre com o valor de c:");
    // resto do programa entra aqui
end
  
```

24

Programa: equação de segundo grau – 3ª versão

```
// corresponde ao resto do programa
delta = b^2 - 4*a*c;
if (delta < 0) then
    printf ("Não existem raízes reais.\n");
else
    if (delta == 0) then
        x = (-b)/(2*a);
        printf ("Há apenas uma raiz: %g", x);
    else
        x1 = (-b + sqrt(delta))/(2*a);
        x2 = (-b - sqrt(delta))/(2*a);
        printf ("As raízes são %g e %g", x1, x2);
    end
end
end
```

25

Operadores Relacionais

>	maior que
>=	maior ou igual a
<	menor que
<=	menor ou igual a
==	igual a
<> ou ~=	diferente de

26

Operadores e valores lógicos

Operador	Notação Scilab
NOT	~
AND	&
OR	

Valores lógicos:

- Verdadeiro: constante lógica %t
- Falso: constante lógica %f

27

Exemplos de operações lógicas

```
-->a = %t; b = %f;
-->~a
ans =
F
-->a & b
ans =
F
-->a | b
ans =
T
-->x = 10; y = 15;
-->a = x > y
a =
F
```

O ";" permite colocar dois comandos em uma mesma linha

28

Comando de repetição **while**

```
while <condição>
    <bloco de repetição>
end
```

29

Programa: equação de segundo grau – 4ª versão

```
// Cálculo das raízes de uma equação
// de 2o grau

// Entrada e validação do coeficiente a,
// forçando-o a ter um valor válido
a = input ("Entre com o valor de a: ");
while (a == 0)
    printf ("O coeficiente a deve ser
    diferente de 0.\n");
    a = input ("Entre com o valor de a: ");
end

// Entrada dos coeficientes b e c
b = input ("Entre com o valor de b: ");
c = input ("Entre com o valor de c: ");
// resto do programa entra aqui
```

30

Comando de repetição **while**

Quando este *loop* vai parar?

```
x = 5
while (x < 10)
    printf("\nx = %g", x);
    x = x - 1;
end
```

Cuidado com os *loops* infinitos!

31

Comando de repetição **for**

```
for <variável> = <inicial>:<final>
    <bloco de repetição>
end
```

```
for <variável> = <inicial>:<passo>:<final>
    <bloco de repetição>
end
```

32

Comando **for** com passo 1

```
for i = 1:5
    printf("\ni = %g", i);
end
```

A variável de controle "i" é incrementada de 1 a cada interação

Saída

```
i = 1
i = 2
i = 3
i = 4
i = 5
```

33

Comando **for** com passo diferente de 1

```
for i = 1:2:10
    printf('\ni = %g', i);
end
```

i varia de 2 em 2

Saída

```
i = 1
i = 3
i = 5
i = 7
i = 9
```

Repare que i não assumiu o limite superior do loop

34

Comando **for** com passo negativo

```
for i = 20:-2:16
    printf('\ni = %g', i);
end
```

Saída

```
i = 20
i = 18
i = 16
```

35

Comando **for** com controle fracionário

A variável de controle pode assumir valores não inteiros

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

Saída

```
x = 0
x = 0.3
x = 0.6
```

36

Equivalência – comandos **while** e **for**

```
for x = 0:2:10
    <bloco de comandos>
end
```

```
x = 0;
while (x <= 10)
    <bloco de comandos>
    x = x + 2;
end
```

37

Programa: fatorial de n

```
// Leitura e validação de n
n = input("Entre com o valor de n = ");
while (n < 0)
    printf (" O valor de n deve ser maior ou
    igual a 0!");
    n = input("Entre com o valor de n = ");
end

// Cálculo do fatorial de n
fat = 1;
if (n > 1) then
    for i = 2:n
        fat = fat * i;
    end
end

// Impressão do resultado
printf("O fatorial de %g é %g", n, fat);
```

38

Programa: Tabela de senos

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

Parada: $x = 2\pi$

39

Programa: Tabela de senos – 1ª versão

```
// Tabela da função Seno
for x = 0:0.2:2*pi
    printf("%g %g", x, sin(x));
end
```

Saída

```
-->
0 00.2 0.1986690.4 0.3894180.6 0.5646420.8 0.7173561 ...
```

40

Programa: Tabela de senos – 2ª versão

```
// Tabela da função Seno
for x = 0:0.2:2*pi
    printf("\n %g %g", x, sin(x));
end
```

0	0
0.2	0.198669
0.4	0.389418
0.6	0.564642
0.8	0.717356
1	0.841471
1.2	0.932039

Saída

41

Programa: Tabela de senos – 3ª versão

```
// Tabela da função Seno

// Impressão do cabeçalho
printf("\n x      seno(x)");

// Impressão das linhas da tabela
for x = 0:0.2:2*pi
    printf("\n %3.1f %7.4f", x, sin(x));
End
```

42

Saída do programa anterior

x	seno(x)
0.0	0.0000
0.2	0.1987
0.4	0.3894
0.6	0.5646
0.8	0.7174
1.0	0.8415
1.2	0.9320

43

"Indentação"

```
if delta < 0 then
    printf('Raízes complexas!');
else
    r1 = (-b + sqrt(delta))/(2*a);
    r2 = (-b - sqrt(delta))/(2*a);
    printf('r1=%g e r2=%g.', r1, r2);
end
```

Mais legível

```
if delta < 0 then
printf('Raízes complexas!');
else
r1 = (-b + sqrt(delta))/(2*a);
r2 = (-b - sqrt(delta))/(2*a);
printf('r1=%g e r2=%g.', r1, r2);
end
```

Menos legível

44

"Indentação"

- Para o Scilab, os dois programas são absolutamente equivalentes.
- Para nós, a disposição do texto do programa afeta muito a legibilidade.
- Qualquer bloco de comando é mais facilmente identificado com "indentação".
 - Assim, os possíveis fluxos de execução ficam mais claros.

45

Strings

- Até o momento, as variáveis definidas armazenam apenas valores numéricos ou lógicos.
- Variáveis podem armazenar também valores alfanuméricos (cadeias de caracteres) denominados *strings*.

```
-->a = "Programação";
a =
Programação
-->b = " de ";
b =
de
-->c = "Computadores";
c =
Computadores
```

Aspas simples (') e duplas (") são equivalentes

46

Concatenação de strings

- *Strings* podem ser concatenados (justapostos).

```
-->a = "Programação";
-->b = " de ";
-->c = "Computadores";
```

```
-->Disciplina = a + b + c;
Disciplina =
Programação de Computadores
```

Para *strings*, + significa concatenação

47

Strings contendo aspas

- Como já visto, o Scilab usa aspas para reconhecer o começo e o fim de um *string*.
- Como, então, representar *strings* que contêm aspas?

```
-->x = 'String "com aspas"';
!--error 276
Missing operator, comma, or semicolon
```

Fim do *string*?

48

Strings contendo aspas

- Para representar *strings* com aspas, deve-se colocar duas aspas consecutivas na posição desejada.

```
-->x = 'String "com aspas duplas"';
x =
String "com aspas duplas"
-->x = 'String 'com aspas simples'';
x =
String 'com aspas simples'
```

49

Strings de dígitos

- *Strings* formados por dígitos não são valores numéricos.

```
-->format(16)
-->%pi
    %pi =
    3.1415926535898
-->StringPi = "3.1415926535898"
    StringPi =
    3.1415926535898
-->2*%pi
    ans =
    6.2831853071796
-->2*StringPi
    !--error 144
    Undefined operation for the given operands
```

Números passam a ser exibidos com 16 posições

50

Programa: passou - não passou

- Faça um programa em Scilab que:
 - leia o nome de um aluno;
 - leia o total de pontos feitos em uma disciplina pelo aluno;
 - retorne, conforme o caso, uma frase do tipo
 "<aluno>, com <tantos pontos>, você passou!"
 ou
 "<aluno>, com <tantos pontos>, você não passou!"

51

Programa: passou - não passou

```
//Leitura do nome
printf("Escreva o seu nome \"entre aspas\".\n");
nomealuno = input("Nome: ");
//Leitura dos pontos obtidos
printf("\n%s, quantos pontos você teve?\n", ...
    nomealuno);
nota = input("Pontos: ");
//Impressão de mensagem com o resultado
if (nota >= 60) then
    printf("Parabéns, %s." + ...
        "\nTendo feito %g pontos, você foi aprovado.\n\n", ...
        nomealuno, nota);
else
    printf("%s, ainda não foi desta vez." + ...
        "\nCom %g pontos, você não foi aprovado.\n\n ", ...
        nomealuno, nota);
end
```

52

Programa: passou - não passou

Comandos

```
printf("Escreva o seu nome \"entre aspas\".\n");
nomealuno = input("Nome: ");
```

Mudança de linha

Para obter aspas

Efeito

Escreva o seu nome "entre aspas".
Nome: "Fulano"

Bug do Scilab 5.1.1:
O *string* não pode conter acentos ou cedilhas.

53

Programa: passou - não passou

Para imprimir uma variável *string*

"..." indicam ao Scilab que o comando continua na linha seguinte

Comandos

```
printf ("\n%s, quantos pontos você teve?\n", ...
        nomealuno);
nota = input("Pontos: ");
```

Efeito

Fulano, quantos pontos você teve?
Pontos: 47

54

Programa: passou - não passou

Comandos

```
if (nota >= 60) then
    printf("Parabéns, %s." + ...
        "\nTendo feito %g pontos, você foi aprovado.\n\n", ...
        nomealuno, nota);
else
    printf("%s, ainda não foi desta vez." + ...
        "\nCom %g pontos, você não foi aprovado.\n\n ", ...
        nomealuno, nota);
end
```

Efeito

Fulano, ainda não foi desta vez.
Com 47 pontos, você não foi aprovado.

55

Processo de repetição

```
continua = %t;
while continua
    // Comandos quaisquer

    // Decisão sobre a continuação do programa
    decisao = ...
        input("Deseja continuar?(s/n)", "string");
    continua = decisao == "s";
end
printf ("Término da repetição.\n");
```

Parâmetro extra do input que elimina a necessidade de aspas ao entrar com *string*

56

Processo de repetição

```
// Cálculo das raízes de diversas equações de 2o grau
continua = %t;
while continua
    a = input("Digite o valor de a:");
    b = input("Digite o valor de b:");
    c = input("Digite o valor de c:");
    delta = b^2 - 4*a*c;
    if delta >= 0 then
        x1 = (-b+sqrt(delta))/(2*a);
        x2 = (-b-sqrt(delta))/(2*a);
        printf ("As raízes são %g e %g", x1, x2);
    else
        printf ("As raízes são complexas");
    end
    // Decisão de continuação pelo usuário
    decisao = input("Outra equação? (s/n)", "string");
    continua = decisao == "s";
end
Printf ("\nTérmino do programa");
```

57

Comandos aninhados

- Blocos internos a comandos condicionais e comandos de repetição podem conter qualquer tipo de comando, incluindo:
 - comandos de atribuição;
 - comandos de entrada e saída de dados;
 - outros comandos condicionais e de repetição.
- Esta generalidade proporciona uma imensa flexibilidade à programação.

58

Comandos aninhados

- Por exemplo, blocos “então” ou “senão” de *ifs* podem conter qualquer tipo de comando, inclusive outros *ifs*.

```
if <condição 1> then
    // comandos
    if <condição 2> then
        // comandos
    else
        if <condição 3> then
            // comandos
        end
    end
end
end
```

59

Programa: conceitos e notas

- Faça um programa Scilab que:
 - leia o nome e a nota de um aluno em uma determinada disciplina;
 - retorne o conceito correspondente, segundo a tabela:

Resultado	Conceito
90 <= Nota <= 100	A
80 <= Nota < 90	B
70 <= Nota < 80	C
60 <= Nota < 70	D
40 <= Nota < 60	E
0 <= Nota < 40	F

60

Programa: conceitos e notas

```
// leitura e validação dos dados de entrada
if Nota >= 90 then
    Conceito = "A";
else
    if Nota >= 80 then
        Conceito = "B";
    else
        if Nota >= 70 then
            Conceito = "C";
        else
            if Nota >= 60 then
                Conceito = "D";
            else
                if Nota >= 40 then
                    Conceito = "E";
                else
                    Conceito = "F";
                end
            end
        end
    end
end
// apresentação do resultado
```

61

A importância da "indentação"

```
if Nota >= 90 then
    Conceito = 'A';
else
    if Nota >= 80 then
        Conceito = 'B';
    else
        if Nota >= 70 then
            Conceito = 'C';
        else
            if Nota >= 60 then
                Conceito = 'D';
            else
                if Nota >= 40 then
                    Conceito = 'E';
                else
                    Conceito = 'F';
                end
            end
        end
    end
end
end
end
end
```

Mais legível

```
if Nota >= 90 then
    Conceito = 'A';
else
    if Nota >= 80 then
        Conceito = 'B';
    else
        if Nota >= 70 then
            Conceito = 'C';
        else
            if Nota >= 60 then
                Conceito = 'D';
            else
                if Nota >= 40 then
                    Conceito = 'E';
                else
                    Conceito = 'F';
                end
            end
        end
    end
end
end
end
end
```

Menos legível

62

Programa: tabuada

- Faça um programa em Scilab que gere a seguinte tabela de tabuada de multiplicação:

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

63

Programa: tabuada – 1ª versão

```
// Tabuada de multiplicação
for linha = 1:9
    for coluna = 1:9
        printf("%g", linha*coluna);
    end
end
```

Corpo do *loop*
externo: imprime
uma linha

Corpo do *loop*
interno: imprime uma
coluna de uma linha

64

Programa: tabuada

- Ao executar o programa anterior, verifica-se a saída não está legível:

12345678924681012141618369121518212 ...

- É preciso:

- após a impressão de uma linha, mudar de linha com o `\n`;
- dentro de cada linha, imprimir cada valor em um número fixo de colunas.

Programa: tabuada – 2ª versão

```
// Tabuada de multiplicação
for linha = 1:9
    for coluna = 1:9
        printf ("%3g", linha*coluna);
    end
    printf("\n");
end
```

Código de formatação

Fora do loop interno

Arquivos

- Arquivos correspondem a unidades de armazenamento, tipicamente gravados em disco magnético.
- Sistemas operacionais, como *Linux* ou *Windows*, permitem que arquivos sejam criados e recuperados por um nome e pela posição em uma hierarquia de diretórios.
- Em relação ao Scilab, existem alguns tipos de arquivos que podem ser lidos, criados ou modificados.
 - Serão apresentados apenas arquivos ASCII (arquivos legíveis por humanos) que podem ser editados, por exemplo, usando o "Bloco de Notas".

Comandos básicos para uso de arquivos

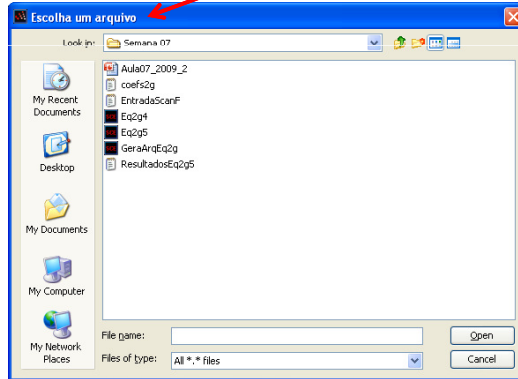
- `uigetfile`
Permite a seleção de um arquivo por meio de "navegação" nos diretórios do *Windows* (ou de outro sistema operacional como o *Linux*).
- `mopen` e `mclose`
Permitem a abertura e o fechamento de arquivos possibilitando, respectivamente, iniciar e finalizar a manipulação dos mesmos.
- `mfscanf`
Permite a leitura de valores contidos em arquivos abertos para variáveis.
- `mfprintf`
Permite a gravação de valores de variáveis em arquivos abertos.
- `meof`
Permite verificar se o fim de um arquivo aberto foi atingido.

Comando **uigetfile**

Diretório cujos arquivos serão apresentados; no caso, **pwd()** indica que a janela deve exibir o diretório corrente do Scilab

```
nomearq = uigetfile("*.*", pwd(), "Escolha um arquivo");
```

Filtro para seleção de arquivos a serem exibidos



69

Comando **uigetfile**

- Após a escolha de um arquivo, a variável **nomearq** recebe como valor um *string* com o nome completo do arquivo.

```
nomearq = C:\Users\Fulano\Ensino\PC1\MeuArquivo.txt
```

- A partir daí, a variável **nomearq** pode ser usada para abrir o arquivo correspondente.
- O nome de arquivo escolhido pode ser novo ou já existir.

70

Comandos **mopen** e **fclose**

- Um arquivo fora de uso está total e tipicamente armazenado em disco.
- Um arquivo em uso tem parte de sua informação em disco e parte em memória principal.
- A abertura de um arquivo, por meio do comando **mopen**, traz para a memória informações necessárias para o seu uso.
- O fechamento de um arquivo, por meio do comando **fclose**, grava em disco todas as informações presentes em memória.

71

Comando **mopen**

Variável **arq** passa a conter um “apontador de arquivo”, a ser usado posteriormente para ler, escrever e fechar o arquivo

```
arq = mopen(NomeCompletoDoArquivo, "r");
```

Variável contendo o nome do arquivo (*string*), muitas vezes obtido por **uigetfile**

Modo de uso do arquivo:

- "r" – leitura
- "w" – escrita

72

Comando **fclose**

```
fclose(arq);
```

Apontador do
arquivo obtido
pelo **mopen**

73

Comando **mfscanf**

Variável que recebe o
número de variáveis
efetivamente lidas em uma
linha do arquivo

String com códigos similares
aos usados em **printf**

```
[n, <lista de variáveis>] = mfscanf(arq, formato);
```

Apontador do arquivo
obtido pelo **mopen**

74

Comando **mfscanf**

- Considere o seguinte arquivo ASCII aberto:

8	32	-40
7	-21	14
5	25	0
7	-63	0

- O comando

```
[n, a, b, c] = mfscanf(arq, "%g %g %g");
```

- em sua primeira execução, faz n=3, a=8, b=32 e c = -40;
- em sua segunda execução, faz n=3, a=7, b=-21 e c=14;
- e assim sucessivamente.

75

Comando **mfprintf**

String com códigos similares
aos usados em **printf**

```
mfprintf(arq, <frase>, <lista de variáveis>);
```

Apontador do arquivo
obtido pelo **mopen**

76

Comando meof

meof (arq)

Função lógica que retorna %t se o fim do arquivo for atingido; caso contrário, retorna %f

Apontador do arquivo obtido pelo **mopen**

■ Uso comum:

```
while ~meof(arq)
// leitura de dados em uma linha do arquivo
[n, a, b, c] = mfprintf (arq, "%g %g %g");
// processamento dos dados da linha lida
end
```

77

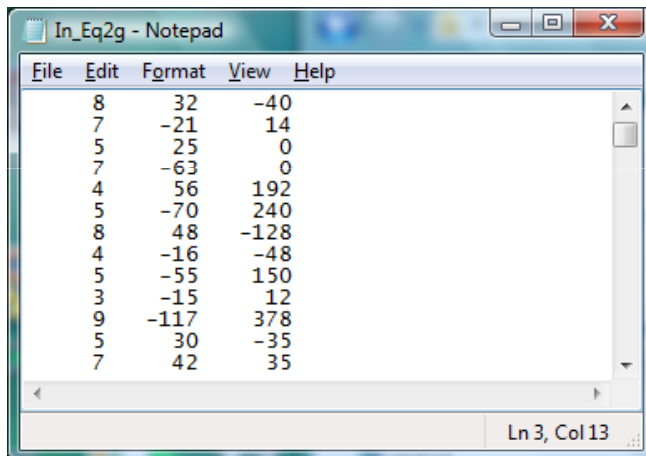
Programa: múltiplas equações de 2º Grau

- Modifique o programa da "equação de 2º grau" de tal forma que se possa calcular as raízes de uma quantidade não determinada de equações de segundo grau, cujos coeficientes estão em um arquivo de entrada.
 - O arquivo possui, em cada linha, os coeficientes **a**, **b** e **c** de uma equação de 2º grau separados por um ou mais espaços.
- O programa deverá produzir um arquivo de saída de tal forma que, em cada linha, devem constar os coeficientes e as raízes reais encontradas de uma equação de 2º grau.
 - Para uma equação de 2º grau cujo delta for negativo, o programa deve gravar os coeficientes e a mensagem "não existem raízes reais" no arquivo de saída.

78

Programa: múltiplas equações de 2º Grau

- As primeiras linhas do arquivo de entrada são:



a	b	c
8	32	-40
7	-21	14
5	25	0
7	-63	0
4	56	192
5	-70	240
8	48	-128
4	-16	-48
5	-55	150
3	-15	12
9	-117	378
5	30	-35
7	42	35

79

Programa: múltiplas equações de 2º Grau

```
// Estrutura geral do programa:

// Localização dos arquivos de entrada e de saída.

// Abertura dos arquivos de entrada e de saída.

// Processamento do arquivo de entrada, envolvendo:
// * leitura dos coeficientes de uma equação;
// * cálculo da equação do 2º grau referente;
// * escrita dos resultados no arquivo de saída.

// Fechamento dos arquivos de entrada e de saída
```

80

Programa: múltiplas equações de 2º Grau

■ Localização dos arquivos de entrada e saída:

```
NomeE = uigetfile("*.txt", pwd(), "Entrada:");
NomeS = uigetfile("*.txt", pwd(), "Saída");
```

Filtro para seleção de arquivos a serem exibidos

`pwd()` indica que a janela deve exibir o diretório corrente do Scilab

81

Programa: múltiplas equações de 2º Grau

■ Abertura dos arquivos de entrada e saída:

```
arqE = mopen(NomeE, "r");
arqS = mopen(NomeS, "w");
```

Modo leitura

Modo escrita

82

Programa: múltiplas equações de 2º Grau

■ Processamento dos arquivos de entrada e saída:

```
while ~feof(arqE)
    [n, a, b, c] = fscanff(arqE, "%g %g %g");
    delta = b^2 - 4*a*c;
    if (delta >= 0) then
        x1 = (-b + sqrt(delta))/(2*a);
        x2 = (-b - sqrt(delta))/(2*a);
        fprintf(arqS, "%8g %8g %8g %8g %8g\n", ...
            a, b, c, x1, x2);
    else
        fprintf(arqS, "%8g %8g %8g %s\n", ...
            a, b, c, "não existem raízes reais");
    end
end
```

83

Programa: múltiplas equações de 2º Grau

■ Fechamento dos arquivos de entrada e saída:

```
fclose(arqE);
fclose(arqS);
```

84

Matrizes

- Matrizes são variáveis que contêm uma quantidade potencialmente grande de valores.
- É no tratamento de matrizes que o Scilab mostra grande superioridade sobre linguagens como C, Fortran ou Java.

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

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

Este comando cria uma matriz 2 x 3, com os valores de cada linha separados por “;”

85

Matrizes

- Todas as variáveis Scilab são, a princípio, matrizes.

```
-->x = 7
x =
7.
```

A função "size" retorna o número de linhas e o número de colunas de uma matriz

```
-->[l,c] = size(x)
c =
1.
l =
1.
```

"x" é uma matriz de uma linha e uma coluna

86

Obtendo o valor de um elemento da matriz

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

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

```
-->e = A(2,3)
e =
6.
```

87

Atribuindo um valor a um elemento da matriz

```
-->A(1,2) = 33
```

1.	33.	3.
4.	5.	6.

88

Vetores

- Vetores são matrizes de uma única linha ou de uma única coluna.

```
-->v = [10 20 30]
```

```
v =
```

```
10.    20.    30.
```

```
-->u = [10; 20; 30]
```

```
u =
```

```
10.
```

```
20.
```

```
30.
```

Para acessar o valor 20,
usa-se "v(1,2)" ou
simplesmente "v(2)"

Para acessar o valor 20,
usa-se "v(2,1)" ou
simplesmente "v(2)"

89

Expansão de uma matriz

- Uma matriz "cresce" quando se atribui valores a elementos ainda não existentes. No caso, as lacunas geradas são completadas com zeros.

```
-->x = 7; // matriz 1x1
```

```
-->x(2,3) = 13
```

```
x =
```

```
7.
```

```
0.
```

```
0.
```

```
0.
```

```
0.
```

```
13
```

90

Atribuindo um valor a uma parte de uma matriz

```
x =
```

23.	30.	29.	50.	91.	28.	68.
23.	93.	56.	43.	4.	12.	15.
21.	21.	48.	26.	48.	77.	69.
88.	31.	33.	63.	26.	21.	84.
65.	36.	59.	40.	41.	11.	40.

```
-->x(2:4, 3:5) = -1
```

```
x =
```

23.	30.	29.	50.	91.	28.	68.
23.	93.	-1.	-1.	-1.	12.	15.
21.	21.	-1.	-1.	-1.	77.	69.
88.	31.	-1.	-1.	-1.	21.	84.
65.	36.	59.	40.	41.	11.	40.

91

Atribuindo valores a uma parte de uma matriz

```
x =
```

40.	58.	38.	73.	53.	4.	58.
87.	68.	92.	26.	11.	67.	48.
11.	89.	94.	49.	22.	20.	22.
19.	50.	34.	26.	62.	39.	84.
56.	34.	37.	52.	76.	83.	12.

```
-->x(3:4, 4:5) = [-1 -2; -3 -4]
```

```
x =
```

40.	58.	38.	73.	53.	4.	58.
87.	68.	92.	26.	11.	67.	48.
11.	89.	94.	-1.	-2.	20.	22.
19.	50.	34.	-3.	-4.	39.	84.
56.	34.	37.	52.	76.	83.	12.

92

Obtendo os valores de uma linha de uma matriz

```
x =
 40.   58.   38.   73.   53.   4.   58.
 87.   68.   92.   26.   11.   67.   48.
 11.   89.   94.   49.   22.   20.   22.
 19.   50.   34.   26.   62.   39.   84.
 56.   34.   37.   52.   76.   83.   12.
```

```
--> a = x(2, :)
```

```
a =
 87.   68.   92.   26.   11.   67.   48.
```

":" designa todos os elementos de uma dimensão (no caso, coluna)

93

Obtendo os valores de colunas de uma matriz

```
x =
 91.   28.   68.   40.   58.   38.   73.
 4.   12.   15.   87.   68.   92.   26.
 48.   77.   69.   11.   89.   94.   49.
 26.   21.   84.   19.   50.   34.   26.
 41.   11.   40.   56.   34.   37.   52.
```

```
--> b = x(:, 3:5)
```

```
b =
 68.   40.   58.
 15.   87.   68.
 69.   11.   89.
 84.   19.   50.
 40.   56.   34.
```

94

Aritmética matricial

- Como todas as variáveis Scilab são matrizes, as operações aritméticas usuais (+, -, *, /, ^) são entendidas pelo Scilab como operações matriciais.

- Assim, **a*b** designa o produto matricial da matriz **a** pela matriz **b**.

- Operações escalares usam os mesmos símbolos aritméticos, porém precedidos por um "." (ponto) como, por exemplo, **.*** e **.^**.

95

Adição e subtração de matrizes

- Matrizes de mesmas dimensões podem ser somadas ou subtraídas.

```
-->x = [1 2 3; 4 5 6];
-->y = [10 20 30; 40 50 60];
-->x + y
ans =
 11.   22.   33.
 44.   55.   66.
-->x - y
ans =
 - 9.   - 18.   - 27.
 - 36.   - 45.   - 54.
```

96

Produto matricial

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

```
x =
```

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

```
-->y = [10 20; 30 40; 50 60]
```

```
y =
```

```
10.    20.
30.    40.
50.    60.
```

```
-->x * y
```

```
ans =
```

```
220.    280.
490.    640.
```

O n° de colunas da 1ª matriz deve ser igual ao n° de linhas da 2ª matriz

$$220 = 1 \times 10 + 2 \times 30 + 3 \times 50$$

97

Produto elemento a elemento de matrizes

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

```
-->y = [10 20; 30 40];
```

```
-->x * y
```

```
ans =
```

```
70.    100.
150.    220.
```

```
-->x .* y
```

```
ans =
```

```
10.    40.
90.    160.
```

Produto matricial

Produto elemento a elemento

98

Multiplicação de matriz por escalar

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

```
-->y = 2*x
```

```
y =
```

```
2.    4.    6.
8.    10.   12.
```

99

Exponenciação em matrizes

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

```
-->x^2
```

```
ans =
```

```
7.    10.
15.    22.
```

```
-->x.^2
```

```
ans =
```

```
1.    4.
9.    16.
```

Produto matricial
 $x * x$

Exponenciação
elemento a elemento

100

Matriz transposta

```

A =
  1.    2.    3.
  4.    5.    6.
  7.   33.    9.

-->B = A'
B =
  1.    4.    7.
  2.    5.   33.
  3.    6.    9.

```

A' é a transposta da matriz A

101

Matriz inversa

```

A =
  4.    7.    6.
  2.    2.    1.
  1.    1.    6.

-->IA = inv(A)
IA =
 - 0.3333333    1.0909091    0.1515152
  0.3333333   - 0.5454545   - 0.2424242
  0.          - 0.0909091    0.1818182

```

A função "inv" retorna a matriz inversa de uma matriz

102

Matriz inversa

```

-->A * IA
ans =
  1.          0.   - 4.441D-16
  1.110D-16   1.   - 1.110D-16
  5.551D-17   0.    1.

-->IA * A
ans =
  1.    8.327D-17    0.
  0.     1.         0.
  0.     0.         1.

```

Erro de aproximação

103

Programa: sistemas de equações lineares

- Um sistema de equações lineares $\mathbf{ax} = \mathbf{b}$ pode ser resolvido pela inversa de uma matriz.
 - Multiplicando os dois lados do sistema por $\mathbf{a}^{-1}$, tem-se:

$$\mathbf{a}^{-1}\mathbf{ax} = \mathbf{x} = \mathbf{a}^{-1}\mathbf{b}$$

- Resolva um sistema de equações lineares, por meio do Scilab, considerando:

$$\mathbf{a} = \begin{bmatrix} -2 & -1 & 3 \\ 2 & 1 & 1 \\ -4 & 1 & 3 \end{bmatrix} \quad \mathbf{b} = \begin{bmatrix} 4 \\ 0 \\ 1 \end{bmatrix}$$

104

Programa: sistemas de equações lineares

```
-->a = [-2 -1 3; 2 1 1;-4 1 3];
-->b = [4 0 1]';
-->x = inv(a)*b

x =
    0.1666667
   -1.3333333
    1.
```

105

Programa: sistemas de equações lineares

- A precisão do resultado calculado pode ser avaliada calculando $\mathbf{ax} - \mathbf{b}$, que deve ser "zero".

```
-->residuo = a*x - b
residuo =
    0.
   1.110D-16
    0.
```

Erro de aproximação

106

Construção de vetores regulares

- Vetores com valores regularmente espaçados podem ser construídos de forma similar à utilizada no comando **for**

```
-->x = 10:13
x =
    10.    11.    12.    13.

-->x = 12:-0.5:10
x =
    12.    11.5    11.    10.5    10.
```

107

Função linspace

- É utilizada para criar um vetor regular especificando seus limites e o número de pontos desejados.

Limite inferior Limite superior Número de pontos

```
-->x = linspace(0,10,3)
x =
    0.    5.   10.

-->x = linspace(0,10,6)
x =
    0.    2.    4.    6.    8.   10.
```

108

Funções zeros e ones

- São utilizadas para criar matrizes com apenas elementos zeros e uns respectivamente. Para tanto, deve-se passar a dimensão desejada da matriz.

```
-->x = zeros(2,3)
x =
    0.    0.    0.
    0.    0.    0.

-->y = ones(2,3)
y =
    1.    1.    1.
    1.    1.    1.
```

109

Função eye

- É utilizada para criar uma matriz identidade. Para tanto, deve-se passar a dimensão desejada da matriz.

```
-->I = eye(4,4)
I =
    1.    0.    0.    0.
    0.    1.    0.    0.
    0.    0.    1.    0.
    0.    0.    0.    1.
```

110

Função rand

- É utilizada para criar uma matriz com elementos aleatórios. Para tanto, deve-se passar a dimensão desejada da matriz.
 - Gera números aleatórios entre 0 e 1.
 - A cada chamada, gera novos números.

```
-->m = rand(2,3)
m =
    0.2113249    0.0002211    0.6653811
    0.7560439    0.3303271    0.6283918

-->n = rand(2,3)
n =
    0.8497452    0.8782165    0.5608486
    0.6857310    0.0683740    0.6623569
```

111

Função rand

Pega a parte inteira
de um número

Fator de escala

```
-->m = int(rand(2,3)*100)
m =
    21.    0.    66.
    75.    33.    62.
```

112

Construindo matrizes a partir de matrizes

```
-->x = [1 2; 3 4];
-->y = [10 20; 30 40];
-->z = [x y]

z =
    1.    2.   10.   20.
    3.    4.   30.   40.

-->z = [x ; y]

z =
    1.    2.
    3.    4.
   10.   20.
   30.   40.
```

113

Funções Scilab são matriciais

- Se uma determinada função for ativada com um argumento matricial, seu resultado será uma matriz.

```
-->x = 0:0.8:%pi
x =
    0.    0.8    1.6    2.4

-->y = sin(x)
y =
    0.    0.7173561    0.9995736    0.6754632
```

114

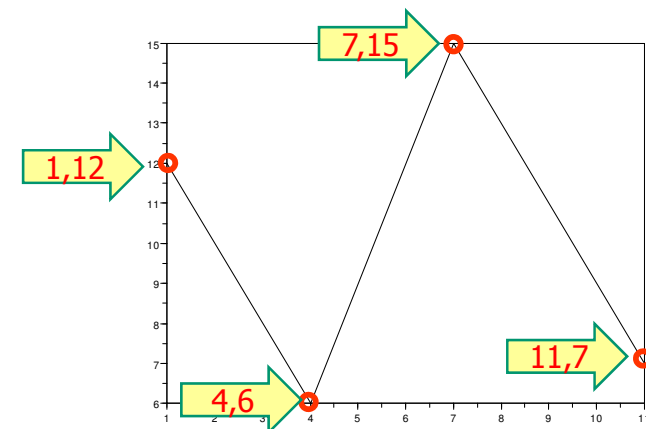
Vetores e gráficos

- Vetores são muito úteis para a construção de gráficos.
- O comando mais simples é `plot2d(x,y)`, onde `x` e `y` são vetores com as mesmas dimensões.
 - Tal comando constrói um gráfico unindo, por retas, os pontos com coordenadas : $(x(1), y(1))$, $(x(2), y(2))$, $(x(3), y(3))$, ...

115

Vetores e gráficos

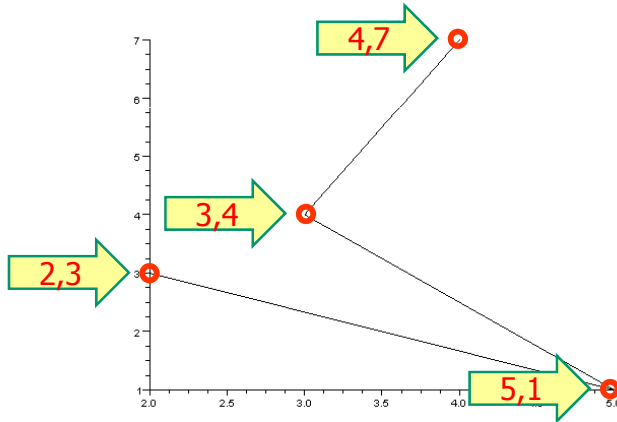
```
-->x = [1 4 7 11]; y = [12 6 15 7];
-->plot2d(x,y)
```



116

Vetores e gráficos

```
-->x = [2 5 3 4]; y = [3 1 4 7];
-->plot2d(x,y)
```

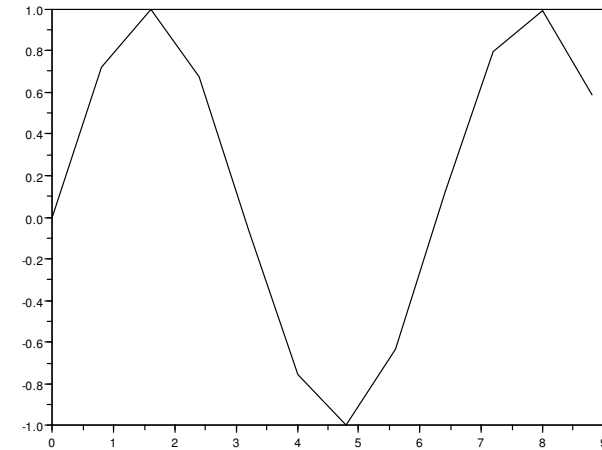


117

Gráfico seno(x) – 1ª Versão

```
-->x = 0:0.8:3*pi;
-->y = sin(x);
-->plot2d(x,y)
```

O espaçamento de 0.8 está grande

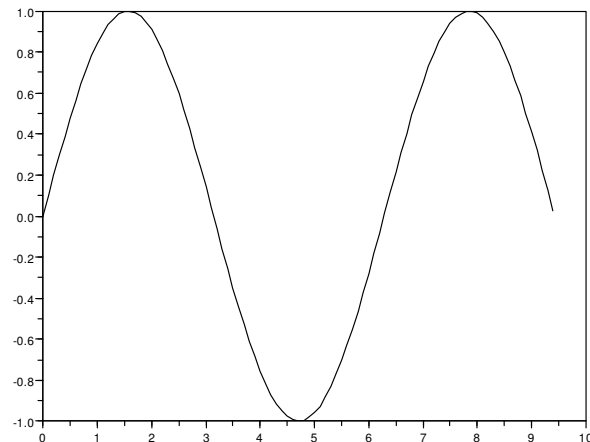


118

Gráfico seno(x) – 2ª Versão

```
-->x = 0:0.1:3*pi;
-->y = sin(x);
-->plot2d(x,y)
```

O espaçamento de 0.1 está bem melhor



119

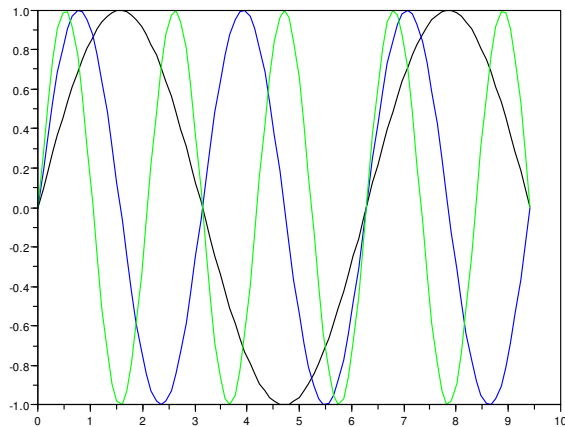
Gráfico com várias curvas

- A função **plot2d** pode ser usada para traçar várias curvas em um único gráfico.
- No caso, **plot2d(x,M)**, onde
 - **x** é um vetor coluna, e
 - **M** é uma matriz com o mesmo número de linhas de **x**, gera um gráfico de **x** versus cada coluna de **M**.

120

Gráfico com várias curvas

```
-->x = linspace(0, 3*pi, 101)';
-->plot2d(x, [sin(x) sin(2*x) sin(3*x)])
```



x é um vetor coluna (e
sin(x), sin(2*x) e
sin(3*x) também são)

121

Matrizes de strings

```
-->a = ["s1" "s2"]
a =

!s1  s2  !

-->b = ["s1" ; "s2"]
b =

!s1  !
!    !
!s2  !
```

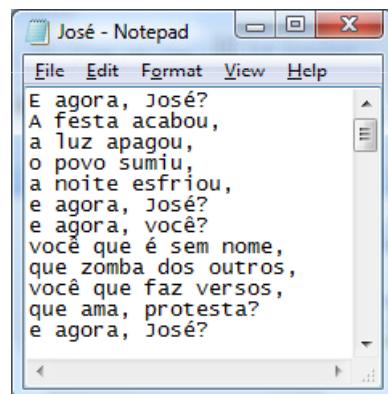
122

Leitura de arquivos como matrizes de strings

- O comando `s = mgetl(arq)`, onde `arq` é o apontador de um arquivo já aberto, lê todas as linhas do arquivo referenciado por `arq` e coloca cada uma delas como um elemento do vetor coluna de strings `s`.

- Exemplificação de uso:

```
fpath = uigetfile();
arq = fopen(fpath,"r");
linhas = mgetl(arq);
fclose(arq);
```



123

Leitura de arquivos como matrizes de strings

- Este programa usado com o arquivo "José" produz:

```
-->linhas
linhas =

!E agora, José?      !
!A festa acabou,    !
!a luz apagou,      !
!o povo sumiu,      !
!a noite esfriou,   !
!e agora, José?     !
!e agora, você?     !
!você que é sem nome, !
!que zomba dos outros, !
!você que faz versos, !
!que ama, protesta?  !
!e agora, José?     !
```

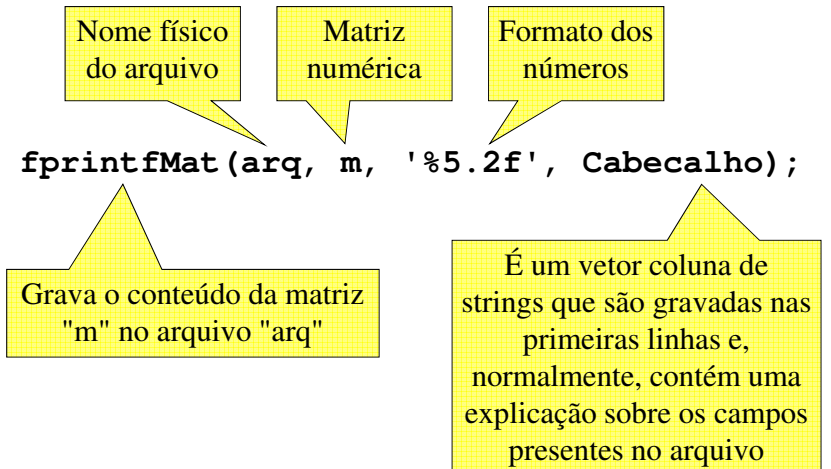
124

Matrizes numéricas e arquivos no Scilab

- Os comandos já vistos de leitura e gravação de arquivos podem ser usados para a leitura de matrizes, mas o Scilab oferece mecanismos mais simples através dos comandos **fscanfMat** e **fprintfMat**.
- Estes comandos leem ou gravam arquivos que contêm somente números em formato tabular, à exceção das primeiras linhas que podem conter textos.
- A abertura e o fechamento dos arquivos são feitas automaticamente.

125

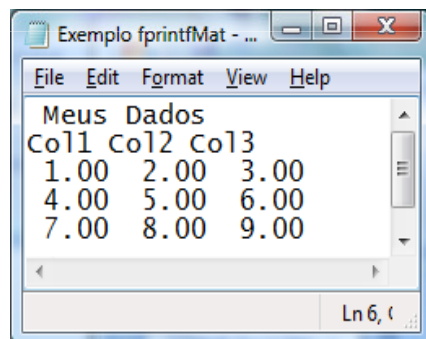
Comando fprintfMat



126

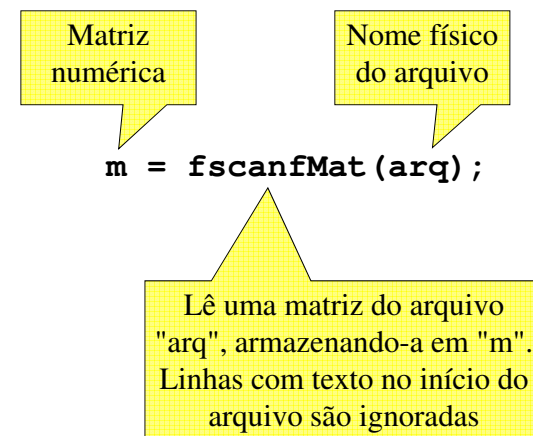
Comando fprintfMat

```
a = [1 2 3; 4 5 6; 7 8 9];
arq = uigetfile();
Cabecalho = [" Meus Dados "; "Col1 Col2 Col3"];
fprintfMat(arq, a, "%5.2f", Cabecalho);
```



127

Comando fscanfMat



128

Comando fscanfMat

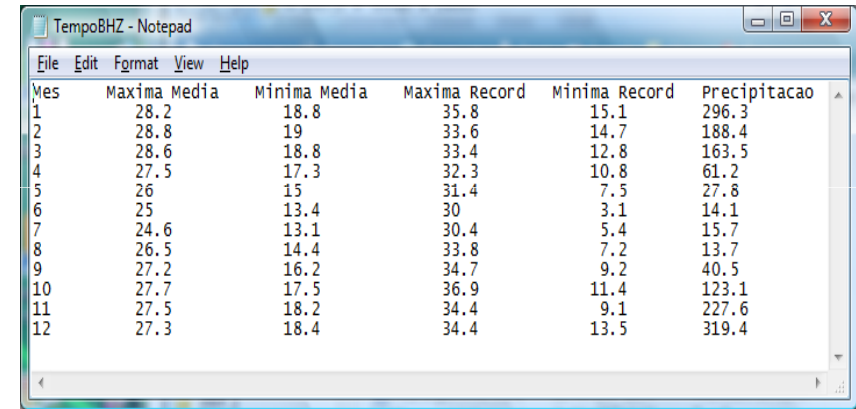
```
arquivo = uigetfile();
m = fscanfMat(arquivo);
```

```
m =

    1.    2.    3.
    4.    5.    6.
    7.    8.    9.
```

129

Programa: Clima em Belo Horizonte



Mes	Maxima Media	Minima Media	Maxima Record	Minima Record	Precipitacao
1	28.2	18.8	35.8	15.1	296.3
2	28.8	19	33.6	14.7	188.4
3	28.6	18.8	33.4	12.8	163.5
4	27.5	17.3	32.3	10.8	61.2
5	26	15	31.4	7.5	27.8
6	25	13.4	30	3.1	14.1
7	24.6	13.1	30.4	5.4	15.7
8	26.5	14.4	33.8	7.2	13.7
9	27.2	16.2	34.7	9.2	40.5
10	27.7	17.5	36.9	11.4	123.1
11	27.5	18.2	34.4	9.1	227.6
12	27.3	18.4	34.4	13.5	319.4

130

Programa: Clima em Belo Horizonte

■ Faça um programa que:

- Leia o arquivo apresentado para uma matriz **ClimaBH**, usando a função **fscanfMat**, que ignora linhas de cabeçalho em um arquivo.
- Da matriz **ClimaBH**, extraia os vetores **MaxMed**, **MinMed**, **MaxRec**, **MinRec** e **Precip**, com significados óbvios.
- Gere um gráfico que tenha simultaneamente os valores de **MaxMed**, **MinMed**, **MaxRec** e **MinRec**.

131

Programa: Clima em Belo Horizonte

```
arqClima = uigetfile();
ClimaBH = fscanfMat(arqClima);

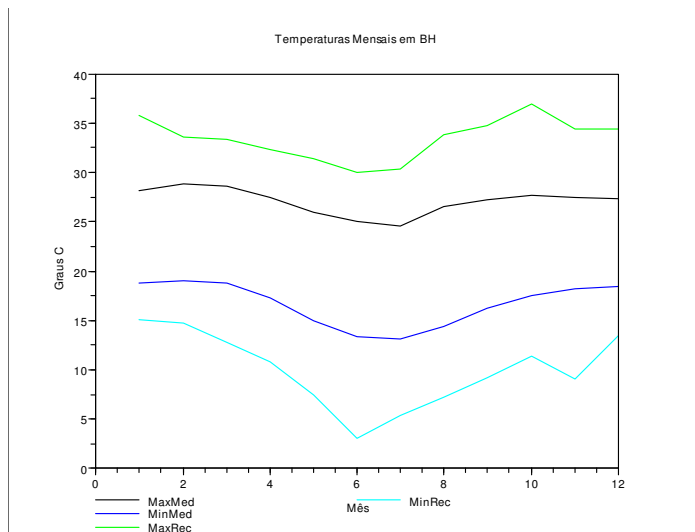
MaxMed = ClimaBH(:,2); // MaxMed = 2a coluna
MinMed = ClimaBH(:,3); // MinMed = 3a coluna
MaxRec = ClimaBH(:,4); // MaxRec = 4a coluna
MinRec = ClimaBH(:,5); // MinRec = 5a coluna
Precip = ClimaBH(:,6); // Precip = 6a coluna

plot2d([1:12], [MaxMed MinMed MaxRec MinRec], ...
leg="MaxMed@MinMed@MaxRec@MinRec");

xtitle("Temperaturas Mensais em BH", "Mês", "Graus C");
```

132

Programa: Clima em Belo Horizonte



133

Matrizes e expressões lógicas

- O resultado de uma expressão lógica envolvendo matrizes é uma matriz de valores lógicos.

<pre>-->a = [3 7; 8 2] a = 3. 7. 8. 2. -->a > 5 ans = F T T F</pre>	<pre>-->a = [3 7; 8 2]; -->b = [5 6; 7 8]; -->a > b ans = F T T F</pre>
--	---

134

Matrizes e expressões lógicas

```
-->a = [3 9; 12 1]
-->x = 0; y = 0;
-->if a > 5 then; x = 10000; end;
-->if a > 0 then; y = 10000; end;
-->[x y]
ans =
    0.    10000.
-->a(a>5) = -1
a =
    3.    - 1.
   - 1.    1.
```

135

Funções

- Funções constituem ferramenta essencial para a modularização de código.
- Vantagens:
 - Permitem reaproveitamento de código.
 - Permitem divisão de tarefas.
 - Tornam código mais legível.

136

Programa: número de combinações

- Faça um programa em Scilab que:
 - leia 2 inteiros n e k ;
 - calcule e apresente o número de combinações de n por k , dado pela fórmula:

$$\binom{n}{k} = \frac{n!}{(n-k)!k!}$$

137

Programa: número de combinações

- Uma das formas de se calcular o fatorial de um número inteiro positivo qualquer é:

```
fat = 1;
for i = 2:n
    fat = fat * i;
end
```

- No caso, o código deve ser adaptado no intuito de se calcular os fatoriais de n , $n-k$ e k .

138

Programa: número de combinações

```
n = input("n= "); k = input("k= ");
fat_n = 1; // Cálculo do fatorial de n
for i = 2:n
    fat_n = fat_n * i;
end
fat_n_k = 1; // Cálculo do fatorial de n-k
for i = 2:(n-k)
    fat_n_k = fat_n_k * i;
end
fat_k = 1; // Cálculo do fatorial de k
for i = 2:k
    fat_k = fat_k * i;
end
nComb = fat_n / (fat_n_k * fat_k);
printf ("Resultado = %g", nComb);
```

139

Programa: número de combinações

Função:

```
function fat = fatorial(n)
    fat = 1;
    for i = 2:n
        fat = fat * i;
    end
endfunction
```

Cabeçalho
da função

Corpo da
função

Programa principal:

```
n = input("n= "); k = input("k= ");
nComb = fatorial(n) / (fatorial(n-k) * fatorial(k));
printf ("Resultado = %g", nComb);
```

Chamada da função

140

Parâmetros formais de uma função

Parâmetro formal de saída,
cujo valor é calculado e
retornado pela função

Parâmetro formal de entrada,
cujo valor deve ser fornecido
na chamada da função

```
function fat = fatorial(n)
    fat = 1;
    for i = 2:n
        fat = fat * i;
    end
endfunction
```

141

Parâmetros formais e reais de uma função

Uma função pode ter mais de
um parâmetro formal de saída

Uma função pode ter mais de
um parâmetro formal de entrada

```
function [r1, r2] = eq2g(a,b,c)
    delta = b^2 - 4*a*c;
    r1 = (-b + sqrt(delta))/(2*a);
    r2 = (-b - sqrt(delta))/(2*a);
endfunction
```

Parâmetros
reais de saída

Chamada da função eq2g :

Parâmetros reais
de entrada

```
[raiz1, raiz2] = eq2g(x,y,z);
```

142

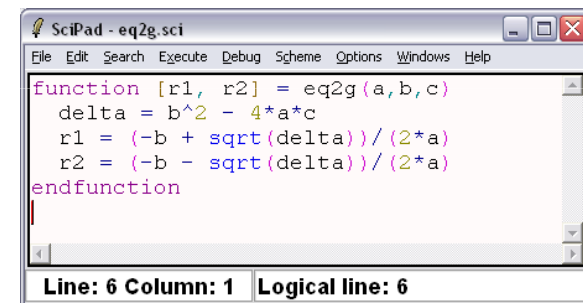
Parâmetros formais e reais de uma função

- Os parâmetros formais de entrada recebem os valores dos parâmetros reais de entrada.
 - Assim, o controle é transferido para a função, que trabalha sobre os parâmetros formais.
- Alterações feitas pela função sobre os parâmetros formais de entrada não afetam os parâmetros reais correspondentes.
 - Assim, variáveis criadas pela função não se misturam com variáveis de mesmo nome existentes no programa que chama a função.
- Os parâmetros reais de saída recebem os valores dos parâmetros formais de saída calculados pela função.
 - Assim, o controle é devolvido para o ponto de chamada.

143

Arquivo com uma função

- Uma função é escrita, normalmente, em um arquivo
 - com o mesmo nome da função;
 - com a extensão **.sci** (um programa tem a extensão **.sce**).



- Para utilizar uma função em um programa Scilab, use **exec(<arquivo com a função>)** em tal programa.

144

Comando exec

```
exec("eq2g.sci")
[raiz1,raiz2] = eq2g(x,y,z);
```

- Um programa Scilab só reconhece a existência de uma função criada pelo programador por meio do comando **exec**.
- O arquivo ".sci" com a função deve estar no mesmo diretório do programa que chama a função.

145

Programa principal e funções em um único arquivo

- Uma outra forma de se trabalhar com funções é construir um único arquivo onde funções precedem o programa principal.
 - Solução mais simples, porém dificulta reaproveitamento e manutenção.

```
function fat = fatorial(n)
    fat = 1;
    for i = 2:n
        fat = fat * i;
    end
endfunction

n = input("n= "); k = input("k= ");
nComb = fatorial(n)/(fatorial(n-k)*fatorial(k));
printf("Resultado = %g", nComb);
```

146

Encadeamento de chamadas

```
function nComb = combinacoes(n,k)
    nComb = fatorial(n)/...
        (fatorial(n-k) * fatorial(k));
endfunction
```

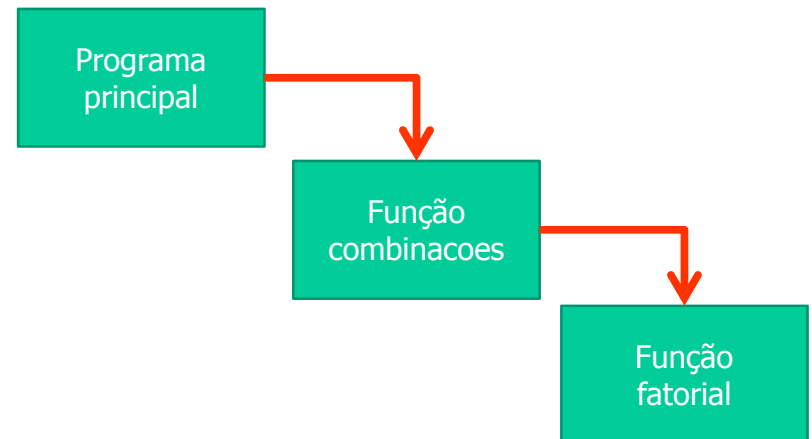
O programa que calcula o número de comparações transformado em função

Programa principal:

```
exec("combinacoes.sci");
exec("fatorial.sci");
n = input("n= "); k = input("k= ");
printf("nComb(%d,%d) = %d",n,k,combinacoes(n,k));
```

147

Encadeamento de chamadas



148

Função: soma dos elementos de um vetor

- Faça uma função para calcular a soma dos elementos de um vetor de valores numéricos.
 - Dados de entrada: um vetor (**parâmetro de entrada**).
 - Dados de saída: soma dos elementos do vetor (**parâmetro de saída**).

```
function s = soma(A)
// Calcula a soma dos
// elementos do vetor A
s = 0;
for k = 1:length(A)
    s = s + A(k);
end
endfunction
```

Função que retorna o número de elementos de um vetor passado como parâmetro

149

Programa principal: teste da função soma

```
exec("soma.sci");

a = int(10*rand(1,4))
sa = soma(a);
printf("\n Soma = %g\n\n", sa);

b = int(10*rand(1,6))
sb = soma(b);
printf("\n Soma = %g\n\n", sb);

c = int(10*rand(1,9))
sc = soma(c);
printf("\n Soma = %g\n\n", sc);
```

A falta do ";" faz com que o vetor seja impresso

150

Execução do programa principal

```
a =
    3.    3.    2.    5.
Soma = 13

b =
    4.    3.    5.    5.    4.    2.
Soma = 23

c =
    6.  4.  9.  0.  4.  2.  4.  2.  1.
Soma = 32
```

151

Função: menor elemento de um vetor

- Faça uma função para retornar o menor elemento de um vetor de valores numéricos.
 - Dados de entrada: um vetor (**parâmetro de entrada**).
 - Dados de saída: menor elemento do vetor (**parâmetro de saída**).

```
function m = menor(A)
// Encontra o menor elemento do vetor A
m = A(1);
for k = 2:length(A)
    if (A(k) < m) then
        m = A(k);
    end
end
endfunction
```

152

Programa principal: teste da função **menor**

```
exec("menor.sci");

a = int(10*rand(1,4));
ma = menor(a);
printf("\n Menor = %g\n\n", ma);

b = int(10*rand(1,6));
mb = menor(b);
printf("\n Menor = %g\n\n", mb);

c = int(10*rand(1,9));
mc = menor(c);
printf("\n Menor = %g\n\n", mc);
```

153

Função: número primo

- Faça uma função para retornar se um determinado número inteiro maior que 1 é primo ou não.
 - Dados de entrada: um número (**parâmetro de entrada**).
 - Dados de saída: valor lógico (**parâmetro de saída**).

```
function primo = ehPrimo(n)
    d = 2;
    while modulo(n,d) ~= 0
        d = d + 1;
    end
    primo = (d == n);
endfunction
```

154

Programa principal: teste da função **ehPrimo**

```
exec("ehPrimo.sci");

for i = 2:100
    if (ehPrimo(i)) then
        printf ("%g é primo\n", i);
    end
end
```

155

Recursividade

- Sabe-se que uma função pode chamar outra função
 - que pode chamar outra função,
 - que pode chamar outra função,
 - e assim sucessivamente ...
- Uma função também pode chamar a si própria.
 - Nesta caso, a função é dita recursiva.
- Pode-se criar uma função recursiva para se resolver um determinado problema quando a definição de tal problema baseia-se nele próprio.

156

Função: fatorial recursivo

- Uma definição recursiva formal de fatorial é:

- $1! = 1$; e
- $n! = n \times (n-1)!$, para $n > 1$.

```
function fat = fatorialR(n)
    if n == 1 then
        fat = 1
    else
        fat = n * fatorialR(n-1)
    end
endfunction
```

157

Programa principal: teste da função **fatorialR**

```
exec("fatorialR.sci");

n = input("n = ");

while n > 0 do
    printf("%d! = %d\n",n,fatorialR(n));
    n = input("n = ");
end
```

158

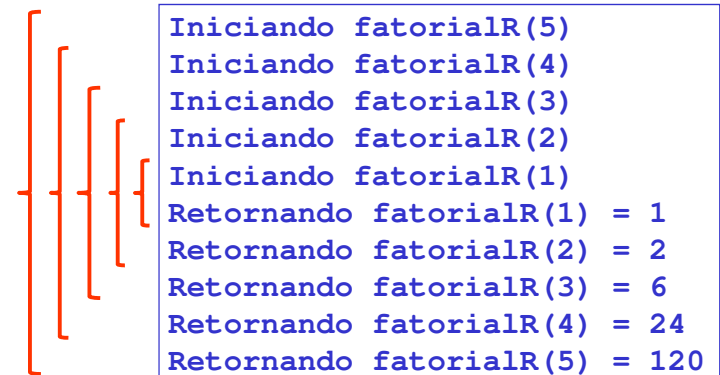
Função: fatorial recursivo com mensagens

```
function fat = fatorialR(n)
    printf("\nIniciando fatorialR(%d)",n);
    if n == 1 then
        fat = 1
    else
        fat = n * fatorialR(n-1)
    end
    printf("\nRetornando fatorialR(%d) = %d",n,fat)
endfunction
```

159

Função: fatorial recursivo com mensagens

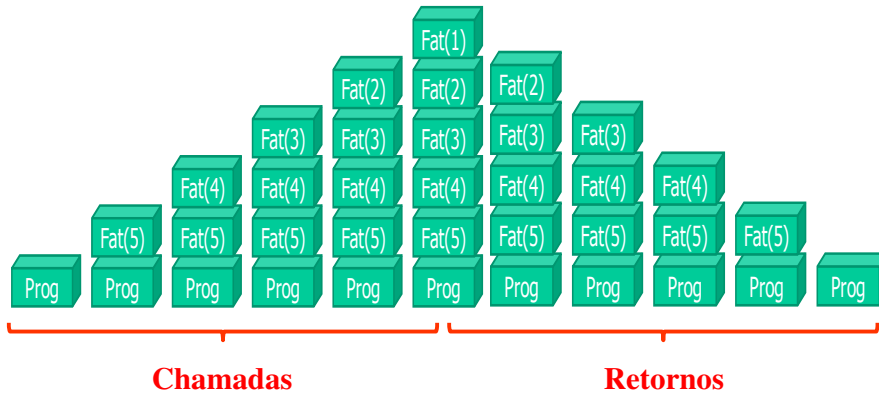
- Execução de **fatorialR(5)**:



```
Iniciando fatorialR(5)
Iniciando fatorialR(4)
Iniciando fatorialR(3)
Iniciando fatorialR(2)
Iniciando fatorialR(1)
Retornando fatorialR(1) = 1
Retornando fatorialR(2) = 2
Retornando fatorialR(3) = 6
Retornando fatorialR(4) = 24
Retornando fatorialR(5) = 120
```

160

Pilha de execução - chamadas e retornos



161

Função recursiva: menor elemento de um vetor

- É possível formular o algoritmo de descoberta do menor elemento em um vetor como uma função recursiva.
- Uma possibilidade é:
 - Se `length(A) == 1`, o menor valor em **A** é **A(1)**.
 - Se `length(A) > 1`, o menor valor em **A** é o menor dentre (o menor valor na metade esquerda de **A**) e (o menor valor na metade direita de **A**).

162

Função recursiva: menor elemento de um vetor

```
function m = menorR(A)
    if (length(A) == 1) then
        m = A(1);
    else
        metade = int(length(A)/2);
        menorEsq = menorR(A(1:metade));
        menorDir = menorR(A(metade+1:length(A)));
        if (menorEsq <= menorDir) then
            m = menorEsq;
        else
            m = menorDir;
        end
    end
endfunction
```

163