As aulas de 1 a 5 foram elaboradas juntamente com o Prof. Ma To FU (UEM)

Procedimentos

No Maple os programas são feitos em forma de procedimentos. Um procedimento é na verdade uma função que transforma os argumentos inseridos em outros argumentos, conforme foi programado. A estrutura de um procedimento é a seguinte

Nomeproced:=proc(argumentos)

local (variáveis locais)

instruções a serem executadas

end;

Num procedimento podem ser necessários comandos de

repetição e iteração

seleção

Vamos falar rapidamente sobre eles.

Certas situações exigem que uma instrução seja repetida várias vezes. Para isto utilizamos o comando "for". A utilização deste comando segue o esquema

(for-do-od)

Por exemplo:

for j from inicio by passo to fim

do expressões dependentes de j

od

A sequencia "do-od"também funciona em outros comandos.

Um outro comando especial para fazer recorrências é o "while" que em português

significa enquanto. O esquema é o seguinte:

while k satisfaz-condição do

instruções envolendo k

od

Os comandos de seleção "if" ou desvio são utilizados para se executar uma instrução

dentre as várias possivies, condicionadas a uma proposição que pode ser falsa ou verdadeira. O esquema é o seguinte:

if condição verdadeira then

instruções a executar

else outras instruçoes a executar

fi

Os camandos de saída de dados são "print"e o "lprint".

A diferença entre eles vai ficar clara nos exemplos.

Exemplo 1. Vamos ver um procedimento para determinar se um dado número é par.

```
> pp:=proc(p)
> if irem(p,2)=0 then
> print(p,`é par`)
> else print(p,`é impar`)
> fi
> end;
```

$$pp := \mathrm{proc}(p)\ \mathrm{if}\ \mathrm{irem}(p, 2) = 0\ \mathrm{then}\ \mathrm{print}(p, `é\ par`)\ \mathrm{else}\ \mathrm{print}(p, `é\ impar`)\ \mathrm{fi}\ \mathrm{end}$$

```
> pp(-3);
```

$$-3, é\ impar$$

```
> pp(6);
```

$$6, é\ par$$

```
> pp(9);
```

$$9, é\ impar$$

```
> pp(2.4);
```

Error, (in pp) wrong number (or type) of parameters in function irem

```
> pp(192837465);
```

$$192837465, é\ impar$$

```
> restart;
```

Exemplo 2. Escrever os primeiros 5 números naturais múltiplos de 7.

```
> m7:=proc(p)
> local k;
> for k from 1 to p do
> print( k*7 );
> od
> end;
```

$$m7 := \mathrm{proc}(p)\ \mathrm{local}\ k;\ \mathrm{for}\ k\ \mathrm{to}\ p\ \mathrm{do}\ \mathrm{print}(7*k)\ \mathrm{od}\ \mathrm{end}$$

```
> m7(5);
```

$$7$$
$$14$$
$$21$$
$$28$$
$$35$$

```
> m7(10);
```

$$7$$
$$14$$
$$21$$
$$28$$
$$35$$
$$42$$
$$49$$
$$56$$
$$63$$
$$70$$

```
> restart;
```

Exemplo3. Os números quadrados perfeitos

```
> k:=0: # k é o contador.
> while k^2<=112
> do
> k:= k+1
> od:
> lprint(`o número de quadrados perfeitos menores que ` ,112,`é`, k);
```

o número de quadrados perfeitos menores que 112 é 11

```
> restart;
```

Exemplo4. As raizes de uma equação do segundo grau

```
> R:=proc(a,b,c)
> if b^2-4*a*c<0 then
> lprint(`as raizes sao complexas`)
> else lprint(` as raizes sao`-b/(2*a)+sqrt(b^2-4*a*c)/(2*a) , -b/(2*a)-sqrt(b^2-4*a*c)/(2*a))
> fi
> end:
>
> R(1,0,1);## solução de x^2+1=0
```

as raizes sao complexas

```
> R(-1,5,6);### solução de -1x^2+5x+6=0
```

` as raizes sao`-1 6

```
>
```

Exemplo 5 . ponto fixo

```
> pfixo1:=proc(f,chute,n)
> local x,k;
> x[0]:=evalf( chute );
> for k from 1 to n do
> x[k]:=evalf( f(x[k-1]) );
> od;
> if abs(x[n]-x[n-1])-abs(x[n-1]-x[n-2])> 0 then ERROR(`a seq diverge`)
```

```
> fi;

> print(evalf(x[n]));

> end;
```

```
pfixo1 := proc(f, chute, n)
local x, k;
    x[0] := evalf(chute);
    for k to n do x[k] := evalf(f(x[k - 1])) od;
    if 0 < abs(x[n] - x[n - 1]) - abs(x[n - 1] - x[n - 2]) then ERROR(`a seq diverge`) fi
    print(evalf(x[n]))
end
```

```
> # Testando

> pfixo1(sqrt, 0.9 ,20);
```

$$.9999998996$$

```
> f:=x->x^3-x;
```

$$f := x \to x^3 - x$$

```
> pfixo1(f,.5,20);
```

$$.1440914959$$

Problema 1. Use o comando print ou lprint para estabelecer algum diálogo com o usuário.

Problema 2. Incluir no programa "pfixo1" um comando de forma a avisar o usuário quando as iterações divergem.

```
> #
```

Exemplo 6. Cálculo de raízes via bissecção.

```
> restart;

> bissec1:=proc(f,a,b,n)

> # argumentos: f=funcao

> # a,b extremos do intervalo [a,b]

> # n=numero de bisseccoes

> local aa, bb, c, k: # variaveis locais

> aa:=a;

> bb:=b;
```

```
> if aa-bb=0 then ERROR(`a`,aa=`b`,bb);
> fi;
> for k from 1 to n do
> c[k]:=evalf( (aa+bb)/2 );
> if evalf( f(aa)*f(c[k]) )>0 then aa:=c[k]
> elif evalf( f(aa)*f(c[k]) )=0 then ERROR(`f(aa)*f(bb)=0`)
> else bb:=c[k]
> fi
> od;
> lprint( `Raiz aproximada apos`,n,`bisseccoes:` );
> lprint( evalf(c[n]) );
> end;
```

```
bissec1 := proc(f, a, b, n)
local aa, bb, c, k;
    aa := a;
    bb := b;
    if aa - bb = 0 then ERROR(a, aa = b, bb) fi;
    for k to n do
        c[k] := evalf(1 / 2*aa + 1 / 2*bb);
        if 0 < evalf(f(aa)*f(c[k])) then aa := c[k]
        elif evalf(f(aa)*f(c[k])) = 0 then ERROR(`f(aa)*f(bb)=0`)
        else bb := c[k]
        fi
    od;
    lprint(`Raiz aproximada apos`, n, `bisseccoes:`);
    lprint(evalf(c[n]))
end
```

```
> #
> # Testando o programa.
> #
> bissec1(sin, 2, 4, 15);
```

Raiz aproximada apos 15 bisseccoes:
3.141540529

```
> #
> ff:= x -> x^3+3*x-10:
```

```
> bissec1(ff, 1, 4, 30);
```

Raiz aproximada apos 30 bisseccoes:
1.698885489

```
> #
```

```
> bissec1(sin, 1, 2, 10); # PERIGO!!! (bug?)
```

Raiz aproximada apos 10 bisseccoes:
1.999023438

Exemplo 7. Cálculo de uma aproximação para $\sqrt{a}$

```
> seqq:=proc(n,chute,m)
> local x,k;
> x[0]:= chute;
> for k from 1 to m do
> x[k]:= evalf((1/2)*(x[k-1]+n/x[k-1]));
> od;
> print(evalf(x[m]));
> end:
```

Exemplos

```
> seqq(2,1,20); # raiz quadrada de 2 apos 20 iteracoes.
```

1.414213563

```
> seqq(5,1,10); # raiz quadrada de 5 apos 10 iteracoes.
```

2.236067978

```
> Digits:= 30:# para pedir uma aproximaçao com 30 digitos
> seqq(2,1,20); # raiz quadrada de 2 apos 20 iteracoes.
```

1.41421356237309504880168872421

```
> seqq(5,1,100); # raiz quadrada de 5 apos 100 iteracoes.
```

2.23606797749978969640917366874

```
>
```