MAPLOVSKY PROGRAMOVACI JAZYK

Zakladni programove konstrukce

for

Prikaz for zajistuje opakovani posloupnosti prikazu v zavislosti na promenne i.
Syntaxe:
for i **from** poc. hodnota **by** krok **to** konecna hodnota **do**
posloupnost prikazu
end do;

```
> for i from 1 by 1 to 5 do
  print(i);
  end do;
```

1
2
3
4
5

Pozn: Pocatecni hodnota, krok, konecna hodnota musi byt vyrazy, jejjichz vycislena hodnota je cele, racionalni cislo nebo FPN.
Je-li vypustena cast from pocatecni hodnota by krok, je automaticky za tyto hodnoty prirazena hodnota 1.

```
> total:=0:
> for i to 5 do
  total:=total+i:
  od;
```

total := 1
total := 3
total := 6
total := 10
total := 15

```
> total;
```

15

Maple v kazde vetvi cyklu zvysuje hodnotu promenne i o jedna a testuje, zda i neni vetsi jak pet. Pokud ne, opakovane provadi prikazy v tele vetve. Soucet se uklada do promenne `total`.

```
> SUM:=proc(n)
  local i, tot;
  tot:=0;
  for i from 1 to n do
  tot:=tot+i;
  end do;
```

```
  tot;
  end:
```

```
> SUM(10);
```

$$55$$

```
> 1+2+3+4+5+6+7+8+9+10;
```

$$55$$

```
> sum('i', 'i'=1..10);
```

$$55$$

```
> printlevel;
```

$$1$$

```
> printlevel:=20;
```

$$printlevel := 20$$

```
> SUM(10);

{--> enter SUM, args = 10
```

$$tot := 0$$
$$tot := 1$$
$$tot := 3$$
$$tot := 6$$
$$tot := 10$$
$$tot := 15$$
$$tot := 21$$
$$tot := 28$$
$$tot := 36$$
$$tot := 45$$
$$tot := 55$$
$$55$$

```
<-- exit SUM (now at top level) = 55}
```

$$55$$

```
> printlevel:=1;
```

$$printlevel := 1$$

```
> SUM:=proc(n)
  local i, tot;
  tot:=0;
  for i from 1 to n do
  tot:=tot+i;
  print('i=',i,'tot=',tot);
```

```
  od;
  tot;
  end:
```

```
> SUM(10);
```

$$i=, 1, tot=, 1$$
$$i=, 2, tot=, 3$$
$$i=, 3, tot=, 6$$
$$i=, 4, tot=, 10$$
$$i=, 5, tot=, 15$$
$$i=, 6, tot=, 21$$
$$i=, 7, tot=, 28$$
$$i=, 8, tot=, 36$$
$$i=, 9, tot=, 45$$
$$i=, 10, tot=, 55$$
$$55$$

```
> SUM:=proc(n::posint)
  local i, tot;
  tot:=0;
  for i from 1 to n do
  tot:=tot+i;
  od;
  tot;
  end:
```

```
> SUM(sqrt(2));

Error, invalid input: SUM expects its 1st argument, n, to be of
type posint, but received 2^(1/2)
```

Nejcasteji pouzivane typy: `array, complex, equation, even, integer, list, name, negint, odd, posint, prime, set, string, evaln.` Dalsi viz ?type.

```
> restart;
  for i from 2 by 2 to 6 do
  Sum(j^i, j=1..n)=expand(sum(j^i, j=1..n));
  end do;
```

$$\sum_{j=1}^{n} j^2 = \frac{1}{3} n^3 + \frac{1}{2} n^2 + \frac{1}{6} n$$

$$\sum_{j=1}^{n} j^4 = \frac{1}{5} n^5 + \frac{1}{2} n^4 + \frac{1}{3} n^3 - \frac{1}{30} n$$

$$\sum_{j=1}^{n} j^6 = \frac{1}{42}n - \frac{1}{6}n^3 + \frac{1}{2}n^5 + \frac{1}{2}n^6 + \frac{1}{7}n^7$$

```
> i;
```

8

Hodnota promenne i je v okamziku ukonceni cyklu pevne dana.
Pr: Vypocet souctu ctvercu sudych cisel z daneho seznamu:

```
> seznam:=[1,2,3,4,5]:
  s:=0:
  for i to nops(seznam) do
  if irem(op(i,seznam),2)=0 then
  s:=s+op(i,seznam)^2
  end if
  end do:

> lprint('soucet ctvercu s =',s);

'soucet ctvercu s =', 20
```

Prikaz for ma jeste dalsi uzitecny tvar:
for x **in** vyraz
do
prikazy uzivajici x
od;

Pr: Ze seznamu se vybiraji jeho cleny a testuji se, zda jsou sude. Pokud ano, tak do promenne s se uklada soucet jejich ctvercu.

```
> s:=0:
  seznam:=[1,2,3,4,5]:
  for n in seznam do
  if irem(n,2)=0 then s:=s+n^2 fi
  od:

> s;
```

20

while

Syntaxe:
while podminka **do** posloupnost prikazu **end do**;
Prikaz while provadi opakovane posloupnosti prikazu tak dlouho, pokud je podminka splnena (tedy pokud je logicky vyraz typu true). Pokud ma podminka hned na zacatku hodnotu false, posloupnost prikazu se neprovede.

```
> x:=256;
```

$x := 256$

```
> while x>1 do x:=x/4 end do;
```

$$x := 64$$
$$x := 16$$
$$x := 4$$
$$x := 1$$

```
> x:=1/2;
```

$$x := \frac{1}{2}$$

```
> while x>1 do x:=x/2 end do;
> x;
```

$$\frac{1}{2}$$

Pr: Nalezeni nejvetsiho spol. delitele dvou celych cisel 20 a 12 pomoci while. (Pouziti Euklidova alg.)

```
> a:=20: b:=12:
  while b<>0 do
   d:=irem(a,b);
   a:=b;
   b:=d;
  end do;
```

$$d := 8$$
$$a := 12$$
$$b := 8$$
$$d := 4$$
$$a := 8$$
$$b := 4$$
$$d := 0$$
$$a := 4$$
$$b := 0$$

```
> lprint('celociselny NSD je',a);

'celociselny NSD je', 4
> igcd(20,12);
```

$$4$$

(kontrola pomoci Maplovske funkce igcd)

```
> euclid:=proc(m::posint,n::posint)
  local a,b,r:
  a:=m:
  b:=n:
  r:=irem(a,b):
```

```
  while r<>0 do
  a:=b:
  b:=r:
  r:=irem(a,b):
  od:
  b:
  end:
```

```
> euclid(20,12);
```

$$4$$

Funkce, ktera vraci nejvetsi mocnimu dvou mensi nebo rovnou n.

```
> binpow:=proc(n::posint)
  local x,m;
  x:=0:
  m:=n:
  while m>=1 do
  m:=m/2:
  x:=x+1;
  od:
  x-1;
  end:
```

```
> for n from 1 to 8 do
  n, binpow(n);
  od;
```

$$1, 0$$
$$2, 1$$
$$3, 1$$
$$4, 2$$
$$5, 2$$
$$6, 2$$
$$7, 2$$
$$8, 3$$

if-then-else-fi

Syntaxe:
if podminka **then** posloupnost prikazu1 **else** posloupnost prikazu2 **end if**;

```
> x:=-2;
```

$$x := -2$$

```
> if x<0 then 0 else 1 end if;
```

0

```
> x:=Pi;
```

$x := \pi$

```
> if eval(x)<0 then 0 else 1 end if;
```

1

```
> x:=1;
```

$x := 1$

```
> if eval(x) < 0 then 0 else 1 end if;
```

1

Vyber z vice moznosti:

if podm1 **then** posl1
elif podm2 **then** posl2
elif podm3 **then** posl3
.
.
elif podmn **then** posln
else posl
fi;

```
> x:=-2;
```

$x := -2$

```
> if x<0 then -1
  elif x=0 then 0
  else 1
  end if;
```

-1

Pr: Je dan matice typu (4x4). Nasledujici prikaz vyplnuje horni trojuhelnikovou matici radkovymi indexy,
cast pod hlavni diagonalou sloupcovymi indexy a hlavni diagonalu jednickami.

```
> A:=array(1..4,1..4):
  for i to 4 do
  for j to 4 do
  if i<j then A[i,j]:=i
  elif i>j then A[i,j]:=j
  else A[i,j]:=1
  end if
  end do end do;
```

```
> evalm(A);
```

$$\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 2 & 2 \\ 1 & 2 & 1 & 3 \\ 1 & 2 & 3 & 1 \end{bmatrix}$$

break

Prikaz break uvnitr cyklu for a while zpusobi bezprostredni ukonceni tohoto cyklu.

Pr: Preruseni cyklu for pomoci prikazu break:
Nalezne prvni hodnotu, ktera neni prvocislo v posloupnosti 2^i-1, i=3,5,7,.. a jakmile ji najde, prerusi vypocet.

```
> for i from 3 by 2 do
  if isprime(2^i-1)
  then print(2^i-1,‘je prvocislo‘)
  else break
  end if
  end do;
```

7, *je prvocislo*

31, *je prvocislo*

127, *je prvocislo*

```
> lprint(‘i=‘,i,‘, prvni neprvocislo v posloupnosti je‘,2^i-1);
```

```
‘i=‘, 9, ‘, prvni neprvocislo v posloupnosti je‘, 511
```

Procedury

Syntaxe:
proc(posloupnost parametru)
local posloupnost promennych;
options posloupnost nabidek;
prikaz1
.
.
.
prikaz n
end proc;

Pozn:
Lokalni promenne - pouzivaji se behem vypoctu uvnitr procedury. **Neovlivnuji hodnotu promenne vne procedury.**
Rozdil mezi lokalnimi a globalnimi promennymi:

```
> restart;
```

```
> b:=2;
```

$$b := 2$$

```
> g:=proc()
  local b;
  b:=5;
  evalf(b/2);
  end:
```

```
> b;
```

$$2$$

```
> g();
```

$$2.500000000$$

```
> b;
```

$$2$$

```
> h:=proc()
  global b;
  b:=5;
  evalf(b/2);
  end:
```

```
> b;
```

$$2$$

```
> h();
```

$$2.500000000$$

```
> b;
```

$$5$$

Obsahuje -li procedura vice prikazu, realizuji se jeden po druhem. Posledni vypocitana hodnota je hodnota procedury.

Pr: Nalezeni maxima ze tri cisel.

```
> max3:=proc(a,b,c)
  print(‘nalezeni maxima z‘, a,b,c);
  if a<b then
   if b<c then c else b end if
   elif a<c then c
   else a
  end if;
  end:
```

```
> max3(3,2,1);
```

nalezeni maxima z, 3, 2, 1

3

max3(3,2,1) provede prikazy v tele procedury tak, ze a ma hodnotu 3, b ma hodnotu 2 a c ma hodnotu 1.

```
> max3(1,8,9);
```

nalezeni maxima z, 1, 8, 9

9

```
> max(1,8,9);
```

9

```
> save(max3, "max3.txt");
> restart;
> read "max3.txt":
> max3(6,3,9);
```

nalezeni maxima z, 6, 3, 9

9

```
> read "sci.txt";
```

sci := **proc**(*a*, *b*) *a* + *b* **end proc**

```
> sci(a,b);
```

a + *b*

Procedury s promennym poctem parametru - uvnitr procedury muzeme ziskat pocet skutecnych parametru pomoci prikazu nargs. Specialni jmeno args ma jako svou hodnotu posloupnost skutecnych parametru. i-ty parametr v teto posloupnosti lze ziskat pomoci args[i].

Pr. Procedura pro hledani maxima z cisel, zadanych jako argumenty. Muze byt zadan libovolny pocet argumentu. procedura testuje, zda jsou vsechny argumenty numericke, ciselne nebo racionalni. Pokud ne, je procedura ukoncena (vraci zadany prikaz).

```
restart;
> maxN:=proc() local result, i;
  if not (type([args], list(numeric)))
   then return('procname(args)');
   elif nargs>0
   then
   result:=args[1];
   for i from 2 to nargs do
                  if args[i]>result then    result:=args[i] fi od;
   result;
  fi;
  end:

> save maxN, "maxN.txt";
```

Specialni jmeno **procname** ma jako svou hodnotu jmeno procedury.

```
> restart;
> read "maxN.txt":
```

```
> maxN(1,2,3);
```

$$3$$

```
> maxN(z,1,2,3);
```

$$\mathrm{maxN}(z, 1, 2, 3)$$

```
> result;i;
```

$$result$$

$$i$$

Prikaz **return** vraci hodnotu procedury.

Lokalni promenne nemaji vne procedury prirazenu hodnotu.
U lokalnich promennych nedochazi k uplnemu vyhodnocovani, vyhodnoceni se deje pouze do urovne 1.
Pokud chceme i u lokalnich promennych dosahnout uplneho vyhodnoceni, musime pouzit funkce eval.

```
> Cfun:=proc()
    local C,y;
    C:=sin(y);
    print(C);
    y:=0;
    print(C);
    print(eval(C));
  end:
```

```
> Cfun();
```

$$\sin(y)$$

$$\sin(y)$$

$$0$$

Pravidla vyhodnocovani procedur
1) Pri volani procedury jsem vsechny jeji parametry plne vyhodnoceny.
Dale jiz nejsou tyto parametry vyhodnocovany.
2) Maple prepise vsechny prikazy procedury a nahradi formalni parametry parametry aktualnimi (vyhodnocenymi).
3) Vsechny lokalni promenne jsou vyhodnoceny pouze o jednu uroven (uvnitr procedury).
Globalni promenne jsou vyhodnocovany uplne (uvnitr procedury).

```
> f:=proc(a,square, invsquare)
> square:=a^2;
> invsquare:=1/square;
> end proc:
>
```

Procedura ma tri vstupni parametry, na vystup jde posledni pocitana hodnota (druhy prikaz v tele procedury). Parametry square a invsquare jsou pouzity pro ukladani vystupu (procedura do nich uklada hodnoty).

```
> A:=h;h:=3;
```

$$A := h$$

$$h := 3$$

```
> f(A,B,C);
```

$$\frac{1}{B}$$

Parametr A se vyhodnoti na 9, B a C se vyhodnoti do sveho jmena. Prikazy procedury se prepisi do tvaru: B:=9; C:=1/b; Po provedeni prikazu je vysledek procedury je 1/B a ne 1/9, jak bychom ocekavali (pravidlo 1). B je vyhodnoceno do jmena pri volani procedury a uz neni znovu vyhodnocovano. Takze i kdyz je prvnim prikazem procedury do B ulozena hodnota 9, C se vyhodnoti do 1/B a uz se dale nevyhodnocuje.

Vsimneme si ale, ze hodnoty B a C jsou nastaveny spravne:

```
> B;C;
```

$$9$$

$$\frac{1}{9}$$

Dalsi volani procedury f(A,B,C) uz by vyvolalo chybove hlaseni:

```
> f(A,B,C);
Error, (in f) illegal use of a formal parameter
```

protoze by vedlo k prirazeni 9:=9; 1/9:=1/9;

```
> restart;
> ff:=proc(a, square, invsquare)
> local sq, isq;
> isq:=1/sq;
> sq:=a^2;
> square:=sq;
> invsquare:=isq;
> end proc:
>
> A:=h;h:=3;
```

$$A := h$$

$$h := 3$$

```
> ff(A,X,Y);
```

$$\frac{1}{sq}$$

Podle pravidla 3) se lokalni promenna isq vyhodnoti do 1/sq (jen o jednu uroven).

```
> X;Y;
```

$$9$$

$$\frac{1}{9}$$

```
> restart;
> fff:=proc(a, square, invsquare)
> square:=a^2;
> invsquare:=1/square;
> square:=1/invsquare;
> end proc;
```

fff := **proc**(*a, square, invsquare*)

square := *a*^2; *invsquare* := 1 / *square*; *square* := 1 / *invsquare*

end proc

```
> fff(A,y,z);
```

$$\frac{1}{z}$$

```
> y;
Error, too many levels of recursion
```

```
> eval(y,1), eval(y,2), eval(y,3), eval(y,4);
```

$$\frac{1}{z}, y, \frac{1}{z}, y$$

```
> restart;
```

Zadani parametru v prubehu procedury (interaktivni vstup) umoznuje funkce readstat.
Pr.

```
> InteractiveDiff:=proc()
  local a,b;
  a:=readstat("Zadej vyraz: ");
  b:=readstat("Derivuj vzhledem k: ");
  print("Derivace je",diff(a,b))end:
```

```
> InteractiveDiff();
Zadej vyraz: y^4;

Derivuj vzhledem k: y;
```

"Derivace je", $4\,y^3$

Ladeni procedur

Podrobneji viz prednaska 12.

```
> restart;
```

Ukladani a nacitani procedur

```
save procedura, "procedura.txt";
save "procedura.m";
read "procedura.txt";
```

```
> CMAX:=proc(x::complex(numeric), y::complex(numeric)) option
  ‘Copyright 2001 R. Plch‘; description "cvicna procedura";if
  abs(x)>abs(y) then
  x;
  else
  y;
  fi;
  end:
```

```
> CMAX(I, 2*I);
```

$$2\,I$$

```
> op(1, eval(CMAX));
```

x::complex(*numeric*), *y*::complex(*numeric*)

posloupnost formalnich parametru

```
> op(2, eval(CMAX));
```

posloupnost lokalnich promennych

```
> op(3, eval(CMAX));
```

Copyright 2001 R. Plch

posloupnost voleb (options)

```
> op(4, eval(CMAX));
```

pametova tabulka

```
> op(5, eval(CMAX));
```

"cvicna procedura"

popis

```
> op(6, eval(CMAX));
```

posloupnost globalnich promennych

```
> save CMAX, "CMAX.txt";
> 
  restart;
```

```
> read "CMAX.txt";
```

CMAX := **proc**(*x*::complex(*numeric*), *y*::complex(*numeric*))
description "cvicna procedura"
...
end proc

```
> CMAX(I,2*I);
```

$$2\,I$$

Vytvareni balicku:
Procedury, ktere spolu nejak souvisi je vhodne ukladat spolecne jako tabulku procedur, kterou pak muzeme jednoduse nacist prikazem **with** (balicek je vlastne tabulka procedur).

```
> 
> 
> powers:=table();
```

powers := table([])

```
> powers[sqr1]:=proc(x)
  x^2;
  end:
```

```
> powers[cube]:=proc(x)
  x^3;
  end:

> powers[sqr1](x+y);
```

$$(x+y)^2$$

```
> with(powers);
```

[*cube, sqr1*]

```
> cube(x);
```

$$x^3$$

```
> save(powers, "powers.m");
> restart;
> libname;
```

"/usr/local/maple95/lib"

```
> libname:= '/home/plch', libname;
```

libname := "/home/plch", "/usr/local/maple95/lib"

```
> with(powers);
```

[*cube, sqr1*]

```
> cube(3);
```

27

Tvorba napovedy

```
> libname:= '/home/plch/maple', libname;
```

libname := "/home/plch/maple", "/home/plch", "/usr/local/maple95/lib"

```
>
>

> makehelp(foo,'pomoc.txt');
> libname;
```

"/home/plch/maple", "/home/plch", "/usr/local/maple95/lib"

```
> ?makehelp
>
```