Promenne v Maplu, vyhodnocovani

Prirazeni a vyhodnoceni

```
> restart;
```

CAS systemy jsou schopny pracovat s formulemi a resit ulohy, ve kterych se vyskytuji nezname a parametry.

```
> solve(a*x^2+b*x+c,x);
```

$$-\frac{b-\sqrt{b^2-4\,a\,c}}{2\,a},-\frac{b+\sqrt{b^2-4\,a\,c}}{2\,a}$$

Zda a,b a c jsou koeficienty a x je neznama. Toto pouziti "volnych promennych", ktere nemaji prirazenu zadnou hodnotu, krome jejich vlastniho jmena, je charakteristicke pro symbolicke systemy.

Promennym prirazujeme hodnoty pomoci operatoru :=.
jmeno:=vyraz;

```
> pol:=9*x^3-37*x^2+47*x-19;
```

$$pol := 9\,x^3-37\,x^2+47\,x-19$$

Urcime koreny tohoto polynomu.

```
> roots(pol);
```

$$\left[[1,\,2],\left[\frac{19}{9},\,1\right]\right]$$

Dosazeni jednoho z korenu do polynomu:

```
> subs(x=19/9,pol);
```

$$0$$

V promenne **pol** je ulozena hodnota 9*x^3-..... Kdykoliv v Maplu zadame promennou pol, Maple pracuje s jeji hodnotou. Tomu rikame vyhodnoceni.
Promenna x stale nema jinou hodnotu (krome sveho jemna), zustava nevyhodnocena.

Prikazem subs jsme v promenne pol nahradili x pomoci 19/9, x zustava nezmeneno. To same plati pro promennou pol.

```
> x;
```

$$x$$

```
> pol;
```

$$9\,x^3-37\,x^2+47\,x-19$$

```
> x:=19/9;
```

$$x := \frac{19}{9}$$

```
> x;
```

$$\frac{19}{9}$$

```
> pol;
```

$$0$$

Maple nejdrive vyhodnoti promennou pol do do 9*x^3-...., potom vyhodnoti kazde x do 19/9 a nakonec provede zjednoduseni.

```
> x+1;
```

$$\frac{28}{9}$$

```
> x:=neznama;
```

$$x := neznama$$

```
> pol;
```

$$9\,neznama^3 - 37\,neznama^2 + 47\,neznama - 19$$

```
> neznama:=7;
```

$$neznama := 7$$

```
> x;
```

$$7$$

```
> pol;
```

$$1584$$

Maple vyhodnocuje tak hluboko, jak je to v danem stavu systemu mozne. Toto obecne pravidlo se nazyva uplne vyhodnoceni.

Vyjimky z pravidla uplneho vyhodnocovani:

vyrazy, ktere jsou uzavreny v pravych uvozovkach se nevyhodnocuji

```
> x;
```

$$7$$

```
> 'x';
```

$$x$$

Uzavreni do pravych uvozovek nezamezuje automaticke zjednoduseni:

```
> 'p+q-i-p+3*q';
```

$$4\,q - i$$

Pokud do pravych uvozovek (apostrofu) uzavreme jednoduchou promennou, je vysledkem jmeno teto promenne. Tohoto muzeme vyuzit k zpetnemu vyhodnoceni promenne pouze do jmena.

```
> x:='x';
```

$$x := x$$

Jmena nalevo od prirazovaciho operatoru := se nevyhodnocuji

```
> x;
```

$$x$$

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

$$x := 1$$

```
> x:=7;
```

$$x := 7$$

```
> i:=1; A[i]:=2; A[i]:=3;
```

$$i := 1$$
$$A_1 := 2$$
$$A_1 := 3$$

Vidime, ze index u indexovane promenne je vyhodnocovan, ale indexovana promenna uz dale vyhodnocovana neni (v tretim prikazu je leva strana **A[i]** vyhodnocena do **A[1]**, ale to uz dale neni vyhodnoceno do 2.

Zpetne vyhodnoceni promenne pouze do jmena s vyuzitim pravych uvozovek nefunguje u indexovanych promennych.

```
> A[i]:='A[i]';
```

$$A_1 := A_i$$

```
> A[i];

Error, too many levels of recursion
```

```
> x:='x';
```

$$x := x$$

```
> x:=x+1;

Error, recursive assignment
```

K odstraneni prirazeni (ukazatele) u indexovanych promennych muzeme pouzit prikaz **evaln()**.

```
> A[i]:=evaln(A[i]); A[i];
```

$$A_1 := A_1$$
$$A_1$$

argument procedury evaln se nevyhodnocuje

tj. stejneho efektu jako prikazem x:='x'; dosahneme i prikazem x:=evaln(x);

```
> evaln(x);
```

$$x$$

```
> restart;
```

```
> p:=q; r:=q*s;
```

$$p := q$$
$$r := q\,s$$

```
> anames();
```

interface, type/interfaceargs, r, p

Vypisuje vsechny promenne s prirazenou hodnotou.

```
> anames('integer');
```

Digits, Order, printlevel

```
> anames('user');
```

r, p

```
> unames():
> nops({%});
```

181

Vypisuje vsechny "volne" promenne.

```
> select(s->length(s)=1, {unames()});
```

{*R, t, e, z, c, A, i, j, L, x, v, q, s, !,* O*, I, n, f, y, B, a, b, k*}

argument procedury assigned se nevyhodnocuje

```
> assigned(r);
```

true

Urcuje, zda je promenna "volna" nebo "vazana".

```
> assigned(q);
```

false

Procedura **assign(name, expression)** ma stejny efekt jako name:=expression s vyjimkou toho, ze prvni argument procedury assig je plne vyhodnocen, zatimco leva strana prirazeni pomoci := neni vyhodnocena.

```
> restart;

> x:=1;
```

$$x := 1$$

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

$$x := 2$$

```
> assign(x,3);

Error, (in assign) invalid arguments
```

Proceduru assign pouzivame pro prirazeni hodnot promennym pri reseni systemu rovnic.

```
> restart;
```

```
> eqns:={x+y=a, b*x-1/3*y=c};
```

$$eqns := \{x + y = a,\ b\,x - \frac{y}{3} = c\}$$

```
> vars:={x,y}:
> sols:=solve(eqns,vars);
```

$$sols := \{y = \frac{3\,(b\,a - c)}{3\,b + 1},\ x = \frac{3\,c + a}{3\,b + 1}\}$$

```
> x;y;
```

$$x$$

$$y$$

```
> assign(sols);
> x, y;
```

$$\frac{3\,c + a}{3\,b + 1},\ \frac{3\,(b\,a - c)}{3\,b + 1}$$

Proceduru (prikaz) **unassign** muzeme pouzit pro odstraneni hodnot u vice promennych naraz.
Vsimneneme si ale opomenuti pravidla uplneho vyhodnocovani:

```
> unassign(x,y): x,y;

Error, (in unassign) cannot unassign '(3*c+a)/(3*b+1)' (argument must be
assignable)
```

$$\frac{3\,c + a}{3\,b + 1},\ \frac{3\,(b\,a - c)}{3\,b + 1}$$

```
> unassign('x','y'): x,y;
```

$$x, y$$

```
> restart;
```

Uplne vyhodnocovani nam dale priblizi nasledujici posloupnost prikazu:

```
> a:=b; b:=c; c:=3;
```

$$a := b$$

$$b := c$$

$$c := 3$$

Pokud ted zadame a;, jaky bude vysledek?

```
> a;
```

$$3$$

Pro zjisteni interni datove reprezentace pouzijeme prikaz **eval**.

```
> eval(a,1);
```

$$b$$

```
> eval(a);
```

$$3$$

```
> eval(b,1);
```

$$c$$

```
> eval(c,1);
```

$$3$$

```
> eval(a,2);
```

$$c$$

```
> eval(a,3);
```

$$3$$

Prikaz **eval** bez doplnujiciho parametru provadi uplne vyhodnoceni argumentu, volitelny parametr urcuje uroven vyhodnoceni.

```
> c:=5;
```

$$c := 5$$

```
> a;
```

$$5$$

```
> eval(a,1);
```

$$b$$

```
> a:=4;
```

$$a := 4$$

```
> a;eval(a,1);
```

$$4$$

$$4$$

```
> restart;
> x:=y: y:=7:
> eval(x,1); x;
```

$$y$$

$$7$$

```
> x:=x:
> y:=9:
```

Jaka hodnota je ted ulozena v promenne x?

```
> x;
```

$$7$$

Nejjistejsi zpusob, jak zamezit vyhodnocovani, je uzavrit argument do apostrofu (’ ’);
Pr:

```
> restart;
> podil:=0;
```

$$podil := 0$$

```
> rem(x^3+x+1,x^2+x+1,x,'podil');
```

$$2 + x$$

```
> podil;
```

$$x - 1$$

Co se stane, pokud zapomeneme apostrofy v predchazejicim prikazu?

```
> podil:=0;
```

$$podil := 0$$

```
> rem(x^3+x+1,x^2+x+1,x,podil);

Error, (in rem) illegal use of a formal parameter
> podil:=x;
```

$$podil := x$$

```
> rem(x^3+x+1,x^2+x+1,x,podil);
```

$$2 + x$$

```
> eval(podil,1), eval(podil,2);
```

$$x, x - 1$$

```
> podil;

Error, too many levels of recursion
> i:=0;
```

$$i := 0$$

```
> sum(ithprime(i), i=1..5);

Error, (in ithprime) argument must be of type posint
> sum(ithprime('i'), i=1..5);

Error, (in sum) summation variable previously assigned, second argument
evaluates to 0 = 1 .. 5
> sum(ithprime('i'), 'i'=1..5);
```


prikaz **seq** nevyhodnocuje svoje argumenty, vsimnete si rozdilneho chovani prikazu **sum**

```
> i:=2;
```

$$i := 2$$

```
> seq(i^2, i=1..5);
```

$$1, 4, 9, 16, 25$$

```
> i;
```

$$2$$

Datove struktury array, table, matrix a proc maji pro vyhodnocovani specialni pravidlo: **last name evaluation.**

```
> restart;
> x:=y;
```

$$x := y$$

```
> y:=z;
```

$$y := z$$

```
> z:=array([[1,2], [3,4]]);
```

$$z := \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$$

```
> x;
```

$$z$$

Maple vyhodnoti x do y, y do z a ukonci vyhodnocovani, protoze posledni promenna z by se vyhodnotila do jedne ze specialnich datovych struktur.
Uplneho vyhodnoceni dosahneme pomoci prikazu eval.

```
> eval(x);
```

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$$

Lokalni promenne uvnitr procedur se vyhodnocuji pouze o jednu uroven. Pokud chceme dosahnout uplneho vyhodnoceni, musime pouzit funkce **eval**.

Jmena promennych

```
> restart;
```

Jmeno promenne muze obsahovat pismena, cislice a podtrzitko, prvnim znakem musi byt pismeno nebo podtrzitko.
Rozlisuje se mezi velkymi a malymi pismeny.
Rezervovana (zakazana) jmena:

```
> ?reserved
```

Vyhnete se dale jmenum, ktera jiz maji v maplu vyznam (cos, sqrt,....) a jmenum, zacinajicim podtrzitkem.

Pokud chcete ve jmenu promenne pouzivat specialnich znaku (;, mezera, atd.), uzavrete jmeno promenne do levych uvozovek:

```
> ‘Odpoved:‘;
```

Odpoved:

```
> Pi:=5;
```

```
Error, attempting to assign to ‘Pi‘ which is protected
```

```
> cos:=x;
```

```
Error, attempting to assign to ‘cos‘ which is protected
```

Pozor na rozdil (viz minule cviceni)!

```
> evalf([Pi, pi]);
```

$[3.141592654, \pi]$

```
> [Pi,pi];
```

$[\pi, \pi]$

```
> [e^x, E^x, exp(x)];
```

$[e^x, E^x, \mathbf{e}^x]$

```
> eval(%, x=0.5);
```

$[e^{0.5}, E^{0.5}, 1.648721271]$

Operator zretezeni (concatenation):

Pouziva se pri generovani (vytvareni) novych jmen promennych (posloupnosti jmen promennych):

```
> X||Y, X||1;
```

XY, *X1*

```
> X || (Y,1);
```

XY, *X1*

```
> X || (1..8);
```

X1, *X2*, *X3*, *X4*, *X5*, *X6*, *X7*, *X8*

```
> ‘‘ || (X, Y) || (1..4);
```

X1, *X2*, *X3*, *X4*, *Y1*, *Y2*, *Y3*, *Y4*

Poznamka: Pokud pouzivame operatoru zretezeni (.), prvni ze jmen se nevyhodnocuje.

```
> a||b;
```

ab

```
> a:=x;
```

$a := x$

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

$b := 2$

```
> a||b;
```

$a2$

```
> c:=3;
```

$c := 3$

```
> a||b||c;
```

$a23$

Vlastnosti promennych

```
> restart;
> (-1)^(m+n);
```

$(-1)^{(m+n)}$

```
> simplify(%);
```

$(-1)^{(m+n)}$

Provedeme nyni zjednoduseni za predpokladu, ze m a n jsou licha.

```
> assume(m::odd,n::odd):
>
> (-1)^(m+n);
```

$(-1)^{(m\sim+n\sim)}$

```
> simplify(%);
```

1

```
> about(m);

Originally m, renamed m~:
  is assumed to be: LinearProp(2,integer,1)
```

```
> additionally (m>0):
> about(m);

Originally m, renamed m~:
  is assumed to be: AndProp(RealRange(Open(0),infinity),LinearProp(2,integer,1)
)
```

Pokud priradime promenne hodnotu, zrusime vsechny predchazejici predpoklady.

```
> m:=3;
```

$m := 3$

```
> about(m);
```

```
3:
  All numeric values are properties as well as objects.
  Their location in the property lattice is obvious,
  in this case integer.
```

Ale:

```
> assume(n::integer);
> expr:=cos(n*Pi);
```

$$expr := (-1)^{n\sim}$$

```
> n:=2;
```

$$n := 2$$

```
> expr;
```

$$(-1)^{n\sim}$$

K cemu doslo? n je prirazena hodnota 2, coz rusi predchazejici predpoklad pro n. Vyraz **expr** stale ukazuje na "starou" promennou n~.

```
> expr:=sqrt(x^2);
```

$$expr := \sqrt{x^2}$$

```
> simplify(expr);
```

$$\mathrm{csgn}(x)\, x$$

```
> assume(x>0):
> simplify(expr);
```

$$x\sim$$

```
> assume(x<0);
> simplify(expr);
```

$$-x\sim$$

```
> coulditbe (x=2);
```

false

```
> coulditbe(x<-1);
```

true

```
> is(x<0);
```

true

Odstraneni vlnky (~) pri vypisu promenne:

```
> interface(showassumed=0);
```

1

```
> assume(q>0);
> q;
```

q~

```
> interface(showassumed=2);
```

0

```
> q;
```

q~

```
> interface(showassumed=1);
```

2

(Implicitni nastaveni.)

```
> q;
```

q~

Datove typy

Jiz jsme se seznamili s typy: integer, fraction, float. Procedura pro urceni datovych typu se jmenuje **whattype**.

```
> whattype(5.0);
```

float

```
> whattype(1);
```

integer

```
> whattype(1/2);
```

fraction

Vysledkem prikazu whattype je popis hlavicky datoveho vektoru.
Prehled ziskame pomoci:

```
> ?surface;
```

K testovani datovych typu se pouziva prikaz **type**.

```
> type(1/2, 'fraction');
```

true

```
> type(1+2*I, complex);
```

true

Narozdil od beznych programovacich jazyku nemusi byt predem deklarovan typ promenne a tento typ se muze behem vypoctu menit.

```
> number:=1: whattype(number);
```

integer

```
> number:=0.75: whattype(number);
```

float

```
> number:=convert(number, 'fraction');
```

$$number := \frac{3}{4}$$

```
> convert(number, 'binary');

Error, invalid input: convert/binary expects its 1st argument, n, to be of type
{float, integer}, but received 3/4
```

```
>
>
```

Posledni priklad ilustruje skutecnost, ze ne vsechny typy zmen jsou v Maplu povoleny. Zmena typu musi davat smysl.