Ladeni programu

```
> restart;
> sit:=proc(n::integer)
   local i,k,flags,count,twice_i;
   count:=0;
   for i from 2 to n do flags[i]:=true od;
   for i from 2 to n do
     if flags[i] then
         twice_i:=2*i;
         for k from twice_i by i to n do
             flags[k]=false;
         od;
         count:=count+l
     fi;
   od;
   count;
  end:
```

Jedna se o Erastothenovo sito. Procedura urcuje pocet prvocisel mensich nebo rovnych zadanemu n. V procedure je zamerne udelano nekolik chyb, ktere mame odstranit.

```
> sit(6);
```

$$5\,l$$

```
> showstat(sit);


sit := proc(n::integer)
local i, k, flags, count, twice_i;
   1   count := 0;
   2   for i from 2 to n do
```

```
   3     flags[i] := true
       end do;
   4   for i from 2 to n do
   5     if flags[i] then
   6       twice_i := 2*i;
   7       for k from twice_i by i to n do
   8         flags[k] = false
           end do;
   9       count := count+l
         end if
       end do;
  10   count
end proc
```

Prikaz **showstat** ocisluje prikazy procedury.

K ladeni potrebujeme spustit procedura a na nekterem miste ji prerusit. Nastaveni bodu preruseni (breakpoint) provedeme prikazem **stopat**.

```
> stopat(sit);
```

[*sit*]

Tento prikaz nastavil preruseni pred prvni prikaz procedury.

```
> sit(10);

sit:
   1*  count := 0;
```

```
DBG> n
10
sit:
   1*  count := 0;
```

```
DBG> next
0
sit:
   2   for i from 2 to n do
```

```
        ...
      end do;
```

```
DBG>
```

Za promptem (vyzvou) rezimu ladeni (**DBG**>) se vypisuje rada informaci:
1) Vysledek predchazejiciho prikazu.
2) Jmeno procedury, jejiz provadeni je zastaveno (zde sit).
3) Cislo prikazu, pred kterym byl beh procedury zastaven, spolu s prikazem.
Nejpouzivanejsim prikazem ladeni je prikaz next, ktery provede zobrazeny prikaz a zastavi beh procedury pred nasledujicim prikazem na stejne urovni.

```
DBG> next
>
true
sit:
   4   for i from 2 to n do
         ...
       end do;
```

Pro skok do vnorene casti procedury pouzijeme prikaz **step**.

```
DBG> step
true
sit:
   5     if flags[i] then
           ...
         end if
```

```
DBG> step
true
sit:
   6       twice_i := 2*i;
```

```
DBG> step
```

```
4
sit:
   7       for k from twice_i by i to n do
             ...
           end do;
```

```
DBG> showstat

sit := proc(n::integer)
local i, k, flags, count, twice_i;
   1*  count := 0;
   2   for i from 2 to n do
   3     flags[i] := true
       end do;
   4   for i from 2 to n do
   5     if flags[i] then
   6       twice_i := 2*i;
   7 !     for k from twice_i by i to n do
   8         flags[k] = false
           end do;
   9       count := count+l
         end if
       end do;
  10   count
end proc
```

! ukazuje prikaz, pri kterem je procedura prerusena.

```
DBG> step
4
sit:
   8         flags[k] = false
```

```
DBG> list

sit := proc(n::integer)
local i, k, flags, count, twice_i;
       ...
   3     flags[i] := true
       end do;
```

```
   4   for i from 2 to n do
   5     if flags[i] then
   6       twice_i := 2*i;
   7       for k from twice_i by i to n do
   8 !       flags[k] = false
           end do;
   9       count := count+l
         end if
       end do;
       ...
end proc
```

Prikaz list ukazuje pouze predchazejici, aktulani a nasledujici prikaz.

```
DBG> outfrom
true = false
sit:
   9       count := count+l
```

Prikaz **outfrom** dokonci vykonavani prikazu na dane urovni vnoreni.

```
DBG> cont
```

9 *l*

Prikaz **cont** dokonci provadeni procedury (pokud nenarazi na dalsi bod preruseni). Vidime, ze procedura nevraci ocekavany vysledek. Pomoci prikazu **unstopat** zrusime bod preruseni.

```
> unstopat(sit);
```

[ ]

Dalsi moznost ladeni predstavuje nastaveni "watchpointu". Watchpoint vyvola rezim ladeni kdykoliv Maple modifikuje nejakou promennou. K nastaveni "sledovacich bodu" pouzijeme prikaz **stopwhen**.

```
> stopwhen([sit,count]);
```

[[*sit*, *count*]]

```
> sit(10);

count := 0
sit:
   2   for i from 2 to n do
         ...
       end do;
```

Beh procedury se prerusi, protoze Maple modifikoval promennou count.

```
DBG> cont
count := l
sit:
   5     if flags[i] then
           ...
         end if
```

```
DBG> cont
count := 2*l
sit:
   5     if flags[i] then
           ...
         end if
```

Misto 2 dostavame 2*l. Ze zdrojoveho textu vidime, ze jsme pouzili l misto 1. Ladeni ukoncime pomoci prikazu **quit**
a opravime zdrojovy text.

```
DBG> quit
Warning, computation interrupted
```

```
> sit:=proc(n::integer)
   local i,k,flags,count,twice_i;
```

```
   count:=0;
   for i from 2 to n do flags[i]:=true od;
   for i from 2 to n do
     if flags[i] then
         twice_i:=2*i;
         for k from twice_i by i to n do
             flags[k]=false;
         od;
         count:=count+1
     fi;
   od;
   count;
  end:
```

```
> unstopwhen();
```

[ ]

```
> sit(10);
```

9

Opet nedostavame spravny vysledek, protoze do desitky mame 4 prvocisla (2,3,5,7). Spustime opet rezim ladeni. Protoze zacatek procedury jsme jiz prosli, nastavime bod preruseni na prikaz cislo 6.

```
> stopat(sit,6);
```

[*sit*]

```
> sit(10);
```

```
true
sit:
   6*      twice_i := 2*i;
```

```
DBG> step
4
sit:
   7       for k from twice_i by i to n do
             ...
           end do;
```

```
DBG> step
4
sit:
   8         flags[k] = false
```

```
DBG> step
true = false
sit:
   8         flags[k] = false
```

Posledni krok ukazuje chybu. Vysledkem posledniho prikazu mela byt hodnota false, ale dostali jsme true=false.
Misto prirazeni jsme zapsali rovnici. Ukoncime ladeni a opet upravime zdrojovy text.

```
DBG> quit
Warning, computation interrupted
```

```
> sit:=proc(n::integer)
   local i,k,flags,count,twice_i;
   count:=0;
   for i from 2 to n do flags[i]:=true od;
   for i from 2 to n do
     if flags[i] then
         twice_i:=2*i;
         for k from twice_i by i to n do
             flags[k]:=false;
         od;
```

```
          count:=count+1
      fi;
    od;
    count;
  end:

> sit(10);
```


Prikazem **DEBUG()** v tele procedury muzeme nastavovat vlastni body preruseni.

```
> f:=proc(x)
  DEBUG("muj bod preruseni, hodnota x
  je:",x);
  x^2
  end:

> f(3);

"muj bod preruseni, hodnota x je:",
3
f:
   2   x^2

DBG> x
3
f:
   2   x^2

DBG> quit
Warning, computation interrupted

> ?debugger
```

Dalsi priklad: Napiste proceduru, ktera pocita hodnoty zadaneho polynomu pro hodnoty z dane mnoziny.

```
> EvalPolyAt:=proc(S)
    local t,p,x,answer;
    p:=x^4-3*x^3-1;
    answer:={};
    for t in S do
      x:=t;
      answer:=answer union {p};
    od;
  RETURN(answer);  end:

> EvalPolyAt({2, 3, 4});

> showstat(EvalPolyAt);

> stopat(EvalPolyAt);

> EvalPolyAt({2,3,4});

DBG> step
DBG> step
DBG> step
DBG> step
DBG> step
DBG> step
DBG> step
DBG> step
DBG> step
DBG> step
> unstopat(EvalPolyAt);
```

```
> EvalPolyAt:=proc(S::set)
   local t,p,x,answer;
   p:=x^4-3*x^3-1;
  answer:=NULL;
  for t in S do
    answer:=answer, subs(x=t,p);
  od;
  RETURN([answer]);
  end:

> EvalPolyAt({2, 3, 4});
```

Chyby

Globalni promenna **lasterror** uchovava posledni chybove hlaseni.

Prikaz **traperror** vyhodnoti svuj argument, pokud nezjisti chybove hlaseni, vraci vyhodnocene argumenty.

```
> x:=0:
> result:=traperror(1/(x+1));
```

$$result := 1$$

Pokud pri vyhodnocovani dojde k chybe, vraci odpovidajici chybove hlaseni.

```
> result:=1/x;

Error, numeric exception: division by zero

> result;
```

$$1$$

```
> result:=traperror(1/x);
```

result := "numeric exception: division by zero"

```
> lasterror;
```

"numeric exception: division by zero"

Srovnanim vysledku procedury traperror s hodnotou promenne lasterror muze testovat, zda doslo k chybe.

```
> evalb(result=lasterror);
```

true

Prikaz **ERROR** ukoncuje proceduru (pouziva se pri testovani, zda jsou zadane parametry pozadovaneho typu a deklarace parametru je pro tento ucel nedostatecna).

```
> restart;
> pairup:=proc(L::list)
  local i,n;
  n:=nops(L);
  if irem(n,2)=1 then
  ERROR( "L musi mit sudy pocet prvku" );
  fi;
  [seq([L[2*i-1],L[2*i]], i=1..n/2)];
  end:

> pairup([1,2,3,4,5]);

Error, (in pairup) L musi mit sudy pocet prvku

> pairup([1,2,3,4,5,6]);
```

$$[[1, 2], [3, 4], [5, 6]]$$

```
> MEMBER:=proc(x,L) local v;
  if not type (L,list) then ERROR("druhy
  argument musi byt seznam") fi;
  for v in L do if v=x then RETURN(true) fi
  od;
  false
  end;
```

MEMBER := **proc**(*x*, *L*)

local *v*;

if not type(*L*, *list*) **then**

ERROR("druhy argument musi byt seznam")

end if;

for *v* **in** *L* **do if** *v* = *x* **then** RETURN(*true*) **end if end do**;

false

end proc

```
> MEMBER(4,5);

Error, (in MEMBER) druhy argument musi byt seznam
```

```
> MEMBER(4,[1,2,3]);
```

false

Ukonceni procedury bez vyhodnoceni.

Hledani maxima ze dvou cisel:

```
> MAX:=proc(x,y) if x > y then x else y fi
  end:
```

Tato procedura ale pracuje pouze s numerickymi hodnotami:

```
> MAX(1,Pi);
```

```
> MAX:=proc(x,y)
  if type (x,numeric) and type(y, numeric)
  then
   if x>y then x else y fi;
  else 'MAX'(x,y);
  fi;
  end:

> MAX(1,Pi);

> MAX:=proc(x,y)
  if type (x,numeric) and type(y, numeric)
  then
   if x>y then x else y fi;
  else if evalf(x) > evalf(y) then x else y
  fi;
  fi;
  end:

> MAX(1,Pi);
```

Ladeni procedury v zavislosti na chybach

Prikaz **stoperror** nastavi preruseni behu procedury v zavislosti na chybovem hlaseni.

Syntaxe: **stoperror("errorMessage"**), pri pouziti paremetru "all" dojde k preruseni behu procedury pri jakekoliv chybe, parametr "traperror" zpusobi preruseni, poku chyba nastane pri

pouziti prikazu traperror.

Prikaz **unstoperror** odstrani prerusovani procedury pri chybach.

```
> restart;
> unstoperror();
```

$$[\,]$$

```
> f:=proc(x) 1/x end:
> g:=proc(x) local r;
  r:=traperror(f(x));
  if r=lasterror then infinity
  else r
  fi
  end:
```

```
> g(9);
```

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

```
> g(0);
```

$$\infty$$

```
> f(0);

Error, (in f) numeric exception: division by zero
```

```
> stoperror(‘numeric exception: division by
  zero‘);
```

["numeric exception: division by zero"]

```
> f(0);
```

```
Error, numeric exception: division by zero
f:
   1   1/x
```

```
DBG> cont
Error, (in f) numeric exception: division by zero
```

Pokud volame f uvnitr g, pouziti traperror zpusobi, ze rezim ladeni se nespousti.

```
> g(0);
```

$$\infty$$

Pouzijme nyni stoperror(traperror).

```
> unstoperror(‘numeric exception: division
  by zero‘);
```

$$[\,]$$

```
> stoperror("traperror");
```

[*traperror*]

Ted se beh procedury neprerusi pri volani f.

```
> f(0);

Error, (in f) numeric exception: division by zero
```

Ale rezim ladeni se spusti pri volani g.

```
> g(0);

Error, numeric exception: division by zero
f:
   1   1/x
```

```
DBG> step
Error, numeric exception: division by zero
g:
   2   if r = lasterror then
         ...
       else
         ...
       end if

DBG> step
Error, numeric exception: division by zero
g:
   3     infinity

DBG> step
                              ∞
> unstoperror();


                             [ ]
>
```