

Процедуры, работа с файлами

1. Функциональные операторы.
2. Общие сведения о процедуре Maple.
3. Вывод описания процедуры на экран.
4. Локальные и глобальные переменные
5. Работа с аргументами процедуры
6. Рекурсивные процедуры.
7. Вывод сообщений об ошибках, выход из процедуры.
8. Работа с файлами.
9. Создание графических приложений Maplets.

§1. Функциональные операторы

Процедура – это подпрограмма, состоящая из команд и выражений Maple. По сути, процедура также является командой Maple, но не встроенной, а созданной пользователем. Упрощенным понятием процедуры является функциональный оператор.

Синтаксис функционального оператора

$$f := \text{var} \rightarrow \text{result}$$
$$f := (\text{var1}, \text{var2}, \dots) \rightarrow \text{result}$$

С помощью функционального оператора можно задавать функции одной и многих переменных, а также вектор-функции:

$f := x \rightarrow x^2$ – функция одной переменной $R \rightarrow R$

$f := (x, y) \rightarrow x^2 + y^2$ – функция двух переменных $R^2 \rightarrow R$

$f := x \rightarrow (2*x, 3*x^4)$ – вектор-функция одной переменной $R \rightarrow R^2$

$f := (x, y, z) \rightarrow (x*y, y*z)$ – вектор-функция трех переменных $R^3 \rightarrow R^2$

Примеры

> g := (x, y) -> sin(x)*cos(y) + x*y; g(Pi/2, Pi);

$g := (x, y) \rightarrow \sin(x) \cos(y) + x y$

Вызов функционального оператора для определенных значений $x = \pi/2$, $y = \pi$;

> g(Pi/2, Pi);

$$-1 + \frac{1}{2} \pi^2$$

С помощью функционального оператора можно создавать небольшие процедуры:

> p:=x-> if x<0 then -x; else sqrt(x); end if: p(-2); p(2);

$$\frac{2}{\sqrt{2}}$$

Создание функционального оператора из выражения и входящих в него переменных

unapply(expression, x, y, ...);
unapply(expression, list of variables);

Примеры

```
> p := x^2 + sin(x) + 1; f := unapply(p,x); f(Pi/6);
```

$$p := x^2 + \sin(x) + 1$$
$$f := x \rightarrow x^2 + \sin(x) + 1$$
$$\frac{1}{36} \pi^2 + \frac{3}{2}$$

```
> q:=a-b;g:=unapply(q,a,b);
```

$$q := a - b$$
$$g := (a, b) \rightarrow a - b$$

```
> q :=x^2+y^3+1; g:=unapply(q,[x,y]); g(2,3);
```

$$q := x^2 + y^3 + 1$$
$$g := (x, y) \rightarrow x^2 + y^3 + 1$$
$$32$$

§2. Общие сведения о процедуре Maple

Процедура – это пользовательская команда Maple. Процедура может не иметь аргументов, иметь один аргумент или иметь несколько аргументов. Если процедура имеет несколько аргументов, то они перечисляются через запятую.

Важно: все описание процедуры должно находиться в одной выполнимой группе (execution group, символ $\left[\begin{smallmatrix} > \\ \end{smallmatrix} \right]$). В этом случае для перехода на новую строку следует нажимать **Shift+Enter**.

Краткий синтаксис процедуры

```
proc_name:=proc (parameterSequence)  
statementSequence;  
end proc;
```

Слова, выделенные **синим**, являются необязательными и могут отсутствовать.

proc_name – имя процедуры

proc ... end proc – служебные слова, которыми должна начинаться и заканчиваться процедура
statementSequence – последовательность выражений, реализующих *тело* процедуры. По умолчанию процедура возвращает значение последнего выражения из этой последовательности.

Пример простейшей процедуры без аргументов

```
> ex := proc (  
     $\sqrt{2}$ ;  
    end proc;  
  
ex := proc (  
     $\sqrt{2}$ ;  
    end proc
```

Вызов и выполнение процедуры:

> *ex*();

$\sqrt{2}$

Полный синтаксис процедуры

```
proc_name := proc (parameterSequence :: type) :: returnType;  
local localSequence;  
global globalSequence;  
option optionSequence;  
description descriptionSequence;  
uses usesSequence;  
statementSequence;  
end proc;
```

Слова, выделенные синим, могут отсутствовать.

parameterSequence — последовательность формальных параметров (*аргументов*) процедуры. Каждому формальному параметру можно предписать (*декларировать*) определенный тип данных с помощью оператора двойного двоеточия :: и следующего за ним названия типа данных *type*. При вызове процедуры в случае несоответствия какого-либо параметра его заявленному типу будет выдаваться системное сообщение об ошибке. Для параметров можно также задать значения по умолчанию.

returnType — необязательный предполагаемый тип возвращаемого значения процедуры. По умолчанию, если тип возвращаемого значения не соответствует предполагаемому, ошибки не происходит.

local — служебное слово для описания последовательности локальных переменных *localSequence*. *Локальными* называются переменные, которые используются только внутри данной процедуры. Для локальных переменных можно задавать тип в виде :: *returnType*.

global — служебное слово для описания последовательности глобальных переменных *globalSequence*. *Глобальными* называются переменные, которые не являются локальными, но также используются данной процедурой. Описание глобальных переменных используется в том случае, если этим переменным внутри процедуры будут присвоены какие-то значения. Для глобальных переменных нельзя задать тип внутри процедуры.

option — служебное слово для описания последовательности опций процедуры *optionSequence*. В качестве опций используются специальные слова, например, **arrow** (стрелка), **builtin** (встроенная процедура), **operator** (оператор), **remember** (опция для эффективной работы рекурсивных процедур), **`Copyright...`** и некоторые другие.

description — служебное слово, за которым следуют комментарии *descriptionSequence* о назначении процедуры и ее работе (одна или несколько строк). В отличие от комментариев, задаваемых символом #, данная информация выводится на экран при печати процедуры.

uses – служебное слово для описания последовательности *usesSequence* связанных имен и модулей, которые будут использованы в теле процедуры. Может быть использовано для подключения пакетов, например: *uses StringTools*;

Пример процедуры с аргументами

```
> p := proc(a, b) a + b; a - b : end proc;
```

Возвращаемым значением процедуры является значение последней команды в теле процедуры.

Кроме того, важен порядок следования аргументов при вызове процедуры.

Вызов и выполнение процедуры (обратите внимание на результат!):

```
> p(1, 2);
```

-1

```
> p(2, 1);
```

1

Параметры, перечисленные в описании последовательности аргументов процедуры, являются обязательными для вызова процедуры. По умолчанию в процедуру передаются все аргументы, содержащиеся в ее вызове, даже если их количество превосходит количество обязательных параметров.

Избыточное количество аргументов при вызове процедуры – ошибки не происходит:

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

-1

Недостаточное количество аргументов при вызове процедуры – выдается сообщение об ошибке:

```
> p(2);
```

Error, invalid input: p uses a 2nd argument, b, which is missing

Пример процедуры с декларированием типов аргументов

```
> f := proc (a :: integer, b) a + b end proc;
```

```
> f(2, 3);
```

5

```
> f(2.5, 3);
```

Error, invalid input: f expects its 1st argument, a, to be of type integer, but received 2.5

Пример процедуры с декларированием нескольких типов данных для аргументов

```
f := proc (a :: {integer, float}, b :: integer) ab end  
proc;
```

Если для параметра требуется задать несколько типов данных, то задается набор данных, например $x :: \{integer, float\}$

```
> f(2, 3);
```

8

```
> f(2, 2.5);
```

Error, invalid input: f expects its 2nd argument, b, to be of type integer, but received 2.5

```
> f(2.5, 2);
```

6.25

Пример процедуры с декларированием типов аргументов и их значений по умолчанию

```
> f := proc (a :: integer := 10, b :: integer
:= 100.1) a + b end proc;
```

Сначала происходит проверка типа аргументов. Вообще говоря, значения по умолчанию для аргументов процедуры могут не соответствовать декларированным типам данных.

Параметры, у которых в описании последовательности аргументов процедуры есть значения по умолчанию, являются необязательными при вызове процедуры.

```
> f(3);
103.1
> f(3, 4);
7
> f(3.5, 4.5)
110.1
```

Пример процедуры с опциями функционального оператора (использование *option*)

```
> f := proc(x) option operator, arrow; x^2-1 end proc;
```

Процедура задает функциональный оператор, ее запись эквивалента команде:

```
> f := x -> x^2-1;
```

Пример процедуры с комментарием о ее назначении (использование *description*)

```
> lc := proc( s, u, t, v )
description "forms a linear combination of the arguments";
s * u + t * v
end proc;
```

Вывод на экран комментариев к процедуре

```
> Describe(lc) ;
# forms a linear combination of the arguments lc( s, u, t, v )
```

Пример процедуры с подключением пакета (использование *uses*)

```
> LastWord:=proc(s::string)
uses StringTools;
Split(s);%[-1];
end proc;
> LastWord("Hello world!");
```

"world!"

```
> LastWord(a) ;
```

Error, invalid input: LastWord expects its 1st argument, s, to be of type string, but received a

Пример. **Возврат нескольких значений**

Процедура находит все простые числа на заданном интервале и выводит их количество и сами числа в виде списка.

```

> PrimesAtInterval := proc(a, b :: integer)
  local COUNT, PRIMES, n;
  COUNT := 0 : PRIMES := [ ];
  for n from a to b do
    if isprime(n) then COUNT := COUNT + 1; PRIMES := [op(PRIMES), n]
    end if
  end do:
  COUNT, PRIMES;
end proc:

```

Обычный вызов

```

> PrimesAtInterval ( 100, 200 );
21, [ 101, 103, 107, 109, 113, 127, 131,
      137, 139, 149, 151, 157, 163, 167,
      173, 179, 181, 191, 193, 197, 199 ]

```

Определение двух возвращаемых значений и их вывод

```

> (n, s) := PrimesAtInterval ( 100, 200 ) :
> n;
21
> s;
[ 101, 103, 107, 109, 113, 127, 131, 137,
  139, 149, 151, 157, 163, 167, 173,
  179, 181, 191, 193, 197, 199 ]

```

Вызов процедуры и вывод первого значения

```

> PrimesAtInterval ( 100, 200 ) [ 1 ];
21

```

Вызов процедуры и вывод второго значения

```

> PrimesAtInterval ( 100 , 200 ) [ 2 ];
[ 101, 103, 107, 109, 113, 127, 131, 137,
  139, 149, 151, 157, 163, 167, 173,
  179, 181, 191, 193, 197, 199 ]

```

§3. Вывод описания процедуры на экран

Вывод описания пользовательской процедуры на экран

```

print(proc_name);
eval(proc_name);

```

Пример пользовательской процедуры

```

> lc := proc( s, u, t, v )
  description "forms a linear combination of the arguments";
  s * u + t * v
end proc:
> eval(lc) ;

```

```

proc(s, u, t, v)
  description
    "form a linear combination of the arguments"
     $s * u + t * v$ 
end proc

```

Вывод описания процедуры из библиотеки Maple на экран

(не работает для встроенных процедур, с опцией builtin)

```

interface('verboseproc'=2): print(proc_name);
interface('verboseproc'=2): eval(proc_name);

```

Пример процедуры из библиотеки

```

> print(issqr) ;

proc(n) ... end proc

> interface('verboseproc' = 2): print(issqr) ;

proc(n)
  option
    Copyright (c) 1990 by the University of
    Waterloo. All rights reserved. ;
  if type(n, integer) then
    evalb(isqrt(n)^2 = n)
  elif type(n, numeric) then
    false
  else
    'issqr(n)'
  end if
end proc

```

Пример встроенной процедуры

```

> interface('verboseproc' = 2): print(conjugate) ;

proc( )
  option builtin = conjugate;

end proc

```

§4. Локальные и глобальные переменные

Пример процедуры с локальными переменными

Процедура *maximum*, находит максимум из заданного списка целых чисел.

```

> maximum := proc (s::(list(integer)))
local max, i;
  max := s[1];
  for i to nops(s) do
    if s[i]>max then

```

```

        max := s[i]
    end if;
end do;
max;
end proc;

> maximum([4,1,8,-100]);

```

8

```

> maximum(4,1,8,-100);
Error, invalid input: maximum expects its 1st argument, s, to be of
type list(integer), but received 4
> maximum([4,1,8,z]);
Error, invalid input: maximum expects its 1st argument, s, to be of
type list(integer), but received [4, 1, 8, z]

```

Если в описании процедуры удалить строку описания локальных переменных, то будут выведены предупреждения о том, что в процедуре используются переменные `max` и `i`, которые будут декларироваться локальными:

```

>
maximum := proc(s :: list(integer))
    max := s[1];
    for i from 1 to nops(s) do
        if s[i] > max then
            max := s[i]
        end if
    end do;
    max;
end proc;

```

```

Warning, `max` is implicitly declared local to procedure `maximum`
Warning, `i` is implicitly declared local to procedure `maximum`

```

Различие между локальными и глобальными переменными

1) Декларирование локальной переменной

```

> my_pi:=3.14:
> CircleArea1:=proc(r)
local my_pi;
my_pi:=evalf(Pi,10);
my_pi*r^2;
end proc:
> CircleArea1(5);

```

78.53981635

```

> my_pi; r:=5: my_pi*r^2;

```

3.14

78.50

2) Декларирование глобальной переменной

```

> my_pi:=3.14:
> CircleArea2:=proc(r)
global my_pi;
my_pi:=evalf(Pi,10);
my_pi*r^2;
end proc:

```



```
> CircleArea2(5);
78.53981635
> my_pi; r:=5: my_pi*r^2;
3.141592654
78.53981635
```

§5. Работа с аргументами процедуры

Для работы с переданными аргументами процедуры есть несколько зарезервированных имен.

_passed – последовательность всех аргументов, переданных процедуре при ее вызове (устаревший вариант: **args**), имеет тип *exprseq*

_npassed – число всех аргументов, переданных процедуре при ее вызове (устаревший вариант: **nargs**)

Пример процедуры с использованием имен _passed и _npassed

Процедура находит максимум из произвольной последовательности чисел.

```
> maximum := proc () local max, i;
    max := _passed[1];
    for i from 2 to _npassed do
        if _passed[i] > max then
            max := _passed[i]
        end if
    end do;
    max;
end proc:
> maximum(2, 5, 77, -10, 100.2);
100.2
```

Если при вызове процедуры число переданных аргументов больше числа обязательных параметров, то доступ к оставшимся «лишним» аргументам можно получить с помощью следующих имен:

_rest – последовательность «лишних» аргументов, переданных процедуре при ее вызове, имеет тип *exprseq*

_nrest – число «лишних» аргументов, переданных процедуре при ее вызове

Пример процедуры с использованием имени _rest

```
> f := proc(a, b) local xarg, x;
    xarg := a + b, _rest;
    for x in xarg do print(x) end do;
end proc:
> f(3, 2);
5
> f(3, 2, 10, 100);
5
```

10
100

> f(3);

Error, invalid input: f uses a 2nd argument, b, which is missing

§6. Рекурсивные процедуры

Рекурсивной называется процедура, которая сама себя вызывает.

Пример рекурсивной процедуры

>

```
myfraction := proc(n)
  local R;
  if n = 1 then R :=  $\frac{1}{1+x}$ ;
  else R :=  $\frac{1}{x + myfraction(n-1)}$ ; end
  if;
end proc;
```

> myfraction(4);

$$\frac{1}{x + \frac{1}{x + \frac{1}{x + \frac{1}{1+x}}}}$$

Использование опции remember для рекурсивных процедур

Пример. Вычисление n-го числа Фибоначчи. Числа Фибоначчи задаются формулой $f_n = f_{n-1} + f_{n-2}$ для $n \geq 2$, $f_0=0$, $f_1=1$.

>

```
Fib1 := proc(n)
  if n < 2 then n
  else Fib1(n-1) + Fib1(n-2)
  end if end proc;
```

>

```
Fib2 := proc(n)
  option remember
  if n < 2 then n
  else Fib2(n-1) + Fib2(n-2)
  end if end proc;
```

> Fib1(30);

832040

> Fib2(30);

832040

Проверим с помощью команды **fibonacci(n)** из пакета **combinat**.

> with(combinat) : fibonacci(30);

832040

Вычислим время работы процедур:

> time(Fib1(30)); time(Fib2(30));

2.324

§5. Вывод сообщений об ошибках, выход из процедуры

Выдача сообщения об ошибке и аварийный выход из процедуры

С помощью команды **error** пользователь может написать для своей процедуры свои сообщения об ошибках. При выполнении команды **error** все оставшиеся команды в теле процедуры игнорируются.

error "Message %1String...%2....",par1,par2,...

В строке сообщения об ошибке вместо %1 подставляется значение *par1*, вместо %2 подставляется значение *par2* и т. д.

Пример

```
sq := proc(x :: numeric)
  if x < 0 then error "Неверный аргумент: %1", x;
  end if;
  sqrt(x);
end proc;
```

Вызов и выполнение процедуры.

> sq(2);

$\sqrt{2}$

> sq(2.5);

1.581138830

> sq(-2);

Error, (in sq) Неверный аргумент: -2

(выдается пользовательское сообщение об ошибке)

> sq('b');

Error, invalid input: sq expects its 1st argument, x, to be of type numeric, but received b

(выдается системное сообщение об ошибке при проверке типа аргумента)

Выход из процедуры в любом месте ее тела и присвоение результату ее работы различных значений

Для выхода из процедуры в любом месте ее тела и возврата значений используется команда **return**. При выполнении команды **return** все оставшиеся команды в теле процедуры игнорируются.

return expr1,expr2,,...

Пример

```
> sq1 := proc(x :: numeric)
  if x < 0 then return abs(x) end if;
  sqrt(x);
end proc;
```

> sq1(4)

2

```
> sql(a)
```

```
Error, invalid input: sql expects its 1st argument, x, to be of type  
numeric, but received a
```

```
> sql(-4.5)
```

4.5

§8. Работа с файлами

См. презентацию к лекции 7, слайды 29-36.

§9. Создание графических приложений Maplets

См. презентацию к лекции 7, слайды 37-51.