15. Процедуры

Во многих языках программирования создание программы облегчают конструкции типа подпрограмм или процедур. Процесс написания программы тогда выражается стратегией "от простого - к сложному", т.е. реализуется принцип "восходящего программирования". Язык Maple V также предоставляет возможность написания процедур, и, таким образом, при необходимости можно запрограммировать достаточно сложные действия, основываясь на уже существующих процедурах. По этому принципу написаны библиотеки Maple, т.е. Maple написан на самом Maple. Исключение составляет лишь ядро среды.

15.1 Определение и вызов процедур

Описать процедуру достаточно просто. Как и большинство конструкций языка синтаксис описания во многом схож с правилами построения выражений языков программирования высокого уровня, таких как C и Pascal.

Синтаксис определения процедуры следующий:
proc (<argseq>)
local <nseq>; global <nseq>; options <nseq>; description <string>;
<statseq>
end;

Параметры:

<argseq>	– список формальных параметров. Параметры в списке разделяются запятой. В списке формальных параметров можно указать через двоеточие тип параметра. Список может быть пустым;
local, global, options	– необязательные параметры, которые определяют глобальные и локальные переменные, а также режимы работы;
description	– комментарий, описание процедуры.

Пример описания и вызова процедуры :

```
> power:=proc(a:integer,b:integer)
> description `возведение в степень`;
>                   a^b
> end;
```

power := proc(*a*::*integer*, *b*::*integer*) description *`возведение в степень` a^b* end

```
> power(2,3); power(2.3,4);
```

8

Error, power expects its 1st argument, a, to be of type integer, but received 2.3

По умолчанию процедура возвращает значение последнего выполненного оператора. Это правило можно изменить при помощи оператора RETURN. Поясним это следующим примером:

```
> sample:=proc(x,y,z)
> local t,c:
> t:=x^2+y^2:
> c:=sqrt(z):
> RETURN(c):
> t*c:
> end:
> sample(2,5,1);
```

1

После описания процедуры ее параметры можно просмотреть при помощи функции op.

op(<num>,eval(<имя процедуры>), где num - число - параметр.

1 - возвращает список формальных параметров
2 - возвращает список локальных переменных
3 - возвращает список ключей из раздела options
4 - возвращает таблицу памяти процедуры
5 - возвращает строку описания description
6 - возвращает список глобальных переменных

Любой возвращаемый список может быть пустым.

15.2 Локальные переменные

Как и в процедурах Pascal, в Maple существуют локальные и глобальные переменные. Они определяются при описании процедуры после ключевых слов global и local. Локальные переменные видимы только в пределах процедуры, глобальные переменные видимы в любом месте рабочего листа.

Рекомендуется все переменные, используемые процедурой, описывать в секциях local и global. Если тип переменных явно не определен, то существует правило определения типа переменной:

• если переменной внутри процедуры присвоено какое-либо значение, то эта пере-

менная считается локальной;
- переменные, используемые в циклах for и seq, также считаются локальными.
- остальные переменные считаются глобальными.

```
> s:=proc(x)
>      y:=2^x;
>      RETURN(y)
> end:
```
Warning, 'y' is implicitly declared local
```
> a:=proc(x)
> global z;
>      x+z
> end:

> z:=0: a(5);
```
5
```
> z:=6: a(5);
```
11

15.3 Параметры описания процедур

Параметры описания процедур записываются в секции options. Параметры приведены в таблице.

Опции описания процедур

ПАРАМЕТРЫ	ОПИСАНИЕ
remember	определяет таблицу памяти для процедуры
builtin	определяет процедуру как "встроенную"
system	определяет "системную" процедуру
operator	объявляет процедуру – функциональный оператор
arrow	уточнение опции operator (стрелочная нотация)
trace	отлаживаемая процедура
package	определяет пакетную процедуру
Copyright	защита от просмотра текста процедуры

Рассмотрим подробно наиболее важные параметры описания процедур.

Параметр remember

Определяет, что все результаты обращений к процедуре будут заноситься в таблицу памяти процедуры.

Рассмотрим пример процедуры, вычисляющей числа Фибоначчи:

```
> f1 := proc(n)
>   if n<2 then n else f1(n - 1)+f1(n - 2) fi
> end:
```

При таком описании процедуры затраты времени на вычисления возрастают по экспоненциальной зависимости.

```
> f2 := proc(n) option remember;
>   if n<2 then n else f2(n - 1)+f2(n - 2) fi
> end:
```

В последнем случае затраты времени возрастают по линейной зависимости, так как предыдущие значения запоминаются в таблице памяти.

Чтобы проверить скорость выполнения вычислений воспользуемся функцией showtime.

```
> readlib(showtime):
> showtime();
 01:= f1(25);
```

75025

time = 35.19, bytes = 6024626

```
 02:= f2(25);
```

75025

time = 0.02, bytes = 72422

```
> showtime:=off;
```

С таблицей памяти можно работать как с обычной таблицей. Допустимы присваивания следующего типа:

```
> f2(0):=0;
```

f2(0) := 0

Параметры operator и arrow

Определяют, что с процедурой можно работать как с оператором.

Ключ arrow дополнительно показывает, что оператор записывается в стрелочной нотации. Ключ arrow изменяет правило определения глобальных и локальных переменных: все нелокальные переменные не будут добавлены в список параметров. arrow используется вместе с operator.

```
> oper1:=proc(x,y)
> options operator,arrow;
>    x^2 - y
> end;
```

$$oper1 := (x, y) \rightarrow x^2 - y$$

```
> oper1(4,1);
```

15

Oper1 эквивалентно записи:

```
> oper1:=(x,y) - >x^2 - y;
```

$$oper1 := (x, y) \rightarrow x^2 - y$$

Ключ trace

Определяет, что при выполнении процедуры будет выводиться отладочная информация.

Не подлежат трассировке процедуры, содержащие функции: assigned, eval, evalhf, evalf, evaln, traperror, seq, userinfo.

```
> pro:=proc(x,y,z)
> local t,k,i;
> options  trace;
>    t:=x*y;
>    k:=y+z;
>    i:=x*z;
>    t+k+i;
> end:

> pro(1,2,3);
{--> enter pro, args = 1, 2, 3
```

$$t := 2$$

$$k := 5$$

$$i := 3$$

10

```
<-- exit pro (now at top level) = 10}
```

10

Аналогичные результаты дает функция trace(<имена процедур>), при этом выйти из режима отладки можно с помощью untrace.

Ключ package

Создает библиотечную процедуру.

Примечание: библиотечные процедуры загружаются оператором with.

```
> # A package defined by a procedure
>  linalg2 := proc() local s,t; option package;
>       t := linalg2(_PackageTable):
>      eval(t[eval(op(procname),1)](args));
>  end:

>  linalg2(_PackageTable) := eval(linalg):
>  with(linalg2,addrow);
```

[*addrow*]

15.4 Присваивание значений параметрам

При описании процедуры указывается список ее формальных параметров. При обращении к процедуре указывается список действительных параметров. В следующем примере в процедуре f формальный параметр - a, действительный параметр - 2. При вызове процедуры действительные параметры присваиваются формальным.

```
> f:=proc(a)
>   a^3
> end:
> f(2);
```

8

При передаче действительных параметров в процедуру проверяется совместимость типов формальных и действительных параметров. В случае их несовпадения генерируется ошибка.

Действительные параметры перед передачей в процедуру предварительно вычис-

ляются.

Количество передаваемых в процедуру действительных параметров может быть не равным количеству формальных параметров (но не меньше количества формальных параметров).

```
> f(2,34,56,7);
```

8

В случае, когда надо вернуть несколько значений, можно использовать прием, приведенный в следующем примере:

```
> dd:=proc(a:integer,x:name)
> if a>5 then x:=true: else x:=false: fi:
> RETURN(a*2):
> end:

> zz:='zz':dd(34,zz);
```

68

```
> zz;
```

true

Для работы с параметрами, переданными в процедуру, можно использовать функции args и nargs. Функция args возвращает последовательность фактических параметров, а функция nargs выдает количество переданных параметров. С помощью этих функций можно описать процедуру в которую можно передавать любое количество значений.

```
> minimum:= proc ()
> local r, i;
>  r := args[1];
>  for i from 2 to nargs do if args[i] < r then r :=
args[i] fi od;
>  RETURN(r)
> end:
> minimum(2,3,-5,0,7,-875,6);
```

-875

15.5 Сообщения об ошибках

Выполнение процедуры можно прервать оператором ERROR. В этом случае будет выдано сообщение “Error, <in имя процедуры> < список параметров >”.

Синтаксис: ERROR(список параметров).

```
> f := proc (x) if x<0 then ERROR(`неверное значение x`,
x) else x^(1/2) fi end:

> f(2);
```

$$\sqrt{2}$$

```
> f( - 2);
```

Error, (in f) неверное значение x, -2

Если вызов процедуры произошел в режиме traperror(..), то список параметров ERROR присваивается переменной lasterror.

```
>  traperror(f( - 2)):
>   if " = lasterror then `произошла ошибка` else
`ошибка не происходила` fi;
```

произошла ошибка

Ключевое слово FAIL предназначено для прерывания процедуры. FAIL используется для сообщения о том, что выражение нельзя вычислить.

```
> h:=proc(t)
>    FAIL
> end:
> h(5);
```

FAIL

15.6 Сохранение процедур в файлах на диске

Записать процедуры в файл можно при помощи команды save:

save <список имен>,<имя файла>

Чтение процедур из файла можно осуществить с помощью команды read:

read <имя файла>

```
> f1:=proc(x,y)
>   x^2+y^2
> end:
> f2:=proc(x,y)
>   x^3+y^3
```

```
> end:
> save f1,f2,profile;
> read profile;
```

$$f1 := \mathrm{proc}(x, y)\ x\hat{}2 + y\hat{}2\ \mathrm{end}$$

$$f2 := \mathrm{proc}(x, y)\ x\hat{}3 + y\hat{}3\ \mathrm{end}$$

Процедуры можно записать в файл во внутреннем формате Maple. Для этого просто надо добавить к имени файла расширение ".m" (в противном случае процедуры будут записаны в текстовом формате).

Maple позволяет определить процедуры, объединенные в библиотеку.

```
> mylib[pr2]:=proc(x) x^2 end: mylib[pr3]:=proc(x) x^3
end:
> with(mylib);
```

[*pr2*, *pr3*]

Записать библиотеку можно при помощи команды save `mylib`. Перед использованием библиотеки ее надо считать командой read и подключить для использования оператором with.

15.7 Отладка процедур

При написании сложных процедур возникает необходимость в проверке правильности работы процедуры и выявлении ошибок. Для этой цели в Maple V release 4 введен отладчик (debugger) - средство для отладки процедур.

В качестве наиболее простого средства отладки можно применить команду *tracelast*. Эта команда выводит стек системы после возникновения ошибки. Аналогичный результат можно получить при установке значения переменной printlevel от трех и выше.

```
> tt:=proc(x) 1/x^2 end:
> tt(0);
```

Error, (in tt) division by zero

```
> tracelast;
```

Error, (in tt) division by zero
executing statement: 1/x^2
tt called with arguments: 0

Более полную информацию о выполняемых операциях внутри процедуры можно получить при использовании отладчика. Отладчик - это особый режим работы системы, который использует свой собственный язык.

В приведенном ниже примере устанавливается точка останова (breakpoint) в про-

цедуре rr , при обращении к этой процедуре включается режим отладки и осуществляется работа процедуры в пошаговом режиме.

```
> rr:=proc(x) local z,s: z:=1/x: s:=10/z: s^2: end:
> stopat(rr);
```

[*rr*]

```
> rr(1);
rr:
   1*    z := 1/x;

DBG> step;
1
rr:
   2     s := 10/z;

DBG> step
10
rr:
   3     s^2
DBG> step
```

100

Узнать про каждую команду отладчика и просмотреть примеры использования можно в справочной системе Maple V release 4.