Условный оператор, оператор цикла, итеративные команды Maple

1. Условные операторы if..then, if..then..else, if..then..elif..then.
2. Операторы цикла for/from, for/in, while, смешанные операторы
3. Итеративные команды Maple.

Всюду далее примеры будут приведены для режима интерфейса Worksheet Mode с текстовым (Text Mode) или математическим (Math Mode) режимом ввода.
Для программирования рекомендуется использовать режим Text Mode.

---

§1. Условный оператор

В Maple можно реализовать базовую структуру программирования «ветвление» с помощью различных видов *условных операторов*.

Виды ветвлений:

- обход (условный оператор если-то)
  if…**then**…**end if**
- разветвление (условный оператор если-то-иначе)
  if…**then**…**else**…**end if**
- множественный выбор (условный оператор если-то-иначе-если…)
  if...**then**…**elif**…**then**…<…> **end if**

В качестве условия должно быть задано логическое выражение, в качестве выражений – одно или несколько выражений Maple. Ветка then выполняется, если результат проверки условия – истина (true). В противном случае (false или FAIL) выполняется ветка else (elif).

Простые формы условного оператора

if *condition* **then** *expressions* **end if;**

Пример. Дана переменная *x*. Реализовать следующий условный оператор **if**…**then**…**end if**: если *x* – логическое выражение, то последовательно вывести на экран два сообщения о том, что *x* – логическое выражение и булево выражение. Проверить работу условного оператора для следующих значений x: 3>2, 3.

Команды вывода сообщений на экран

`print(x)` – команда печатает на экран значение переменной **`x`**
`print("xxx")` – команда печатает на экран строку **`"xxx"`**

`printf("bbb %a cc",x)` – команда печатает на экран строку **`"bbb ... cc"`**, в которой вместо **`%a`** подставляется значение переменной **`x`**.

```
> x := 3 > 2 : print(x);
```

$$2 < 3$$

```
> if type(x, boolean)
    then print("x – логическое выражение");
    printf("%a - булево выражение", x) end if;
```

"x - логическое выражение"

```
2 < 3 - булево выражение
```

```
> x := 3 : print(x);
```

3

```
> if type(x, boolean)
    then print("x – логическое выражение");
    printf("%a - булево выражение", x) end if;
```

if *condition* **then** *expressions1* **else** *expressions2* **end if;**

Пример

```
> a:=3; b:=5;
```

$a := 3$

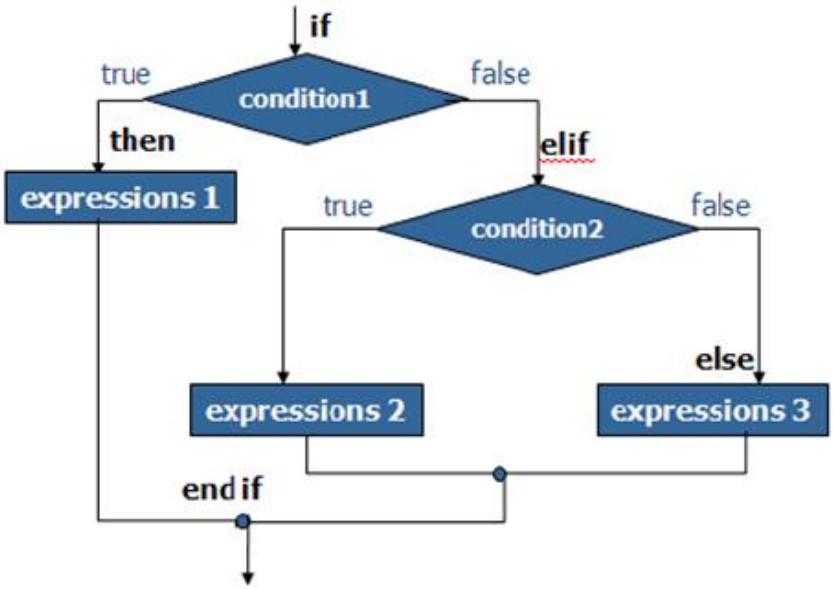
$b := 5$

```
> if (a>b) then a else b end if;
```

5

Полный синтаксис условного оператора

if *condition1* **then** *expressions1* **elif** *condition2* **then** *expressions2* **elif** *condition3* **then** *expressions3* … **else** *expressionsN* **end if;**



Пример

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

```
> if x < 2 then x := x^2; print(x) elif x = 2 then x
    := x^3; print(x) else x := x^4; print(x) end if
```

$x := 2401$

2401

§2. Оператор цикла

В Maple можно реализовать базовую структуру программирования «повторение» с помощью различных видов *операторов цикла* с предусловием. Цикл с постусловием в Maple не реализован. *Выражения*, расположенные между ключевыми словами **do** и **end do** составляют тело цикла.

Виды операторов цикла в Maple

- цикл-пока (цикл while)
  while…do…end do
- циклы с параметром (циклы for/from, for/in)
  for…from…do…end do
  for…in…do…end do
- смешанные циклы (циклы for/from while, for/in while)
  for…from…while…do…end do
  for…in…while…do…end do

Цикл while

while *condition* **do** *expressions* **end do;**

Пока результат проверки условия – истина (true), выполняются выражения Maple, составляющие тело цикла. Как только результат проверки условия становится ложным (false) или FAIL, то происходит выход из структуры цикла.

Пример

```
> j := 0 :
> while j < 5 do j := j + 2; end do;
```

$j := 2$

$j := 4$

$j := 6$

Пример. Четные числа от 8 до 2 в убывающем порядке

```
> j := 8 :
> while j ≥ 2 do if irem(j, 2) = 0 then print(j) end if; j := j − 1 end do;
```

8 — $j := 7$ — 1-я итерация

$j := 6$ — 2-я итерация

6 — $j := 5$ — 3-я итерация

$j := 4$ — 4-я итерация

4 — $j := 3$ — 5-я итерация

$j := 2$ — 6-итерация

2 — $j := 1$ — 7-я итерация

Пример. Частично реализовать алгоритм перевода десятичного числа в двоичное, последовательно выводя остаток и частное от деления на основание двоичной системы. Остатки заносить в последовательность. Для проверки работы алгоритма использовать команду **`convert(x,base,2)`**, выводящую последовательность остатков от деления на 2. Для получения остатка от деления числа x на 2 использовать команду **`irem(x,2)`**. Для получения частного от деления числа x на 2 использовать команду **`iquo(x,2)`**.

```
> x := 19; s := NULL;
```

$$x := 19$$

$$s :=$$

```
> while x>0 do printf("Остаток от деления %a на 2 равен %a", x,
irem(x,2)); s:=s,irem(x,2); printf("Частное от деления %a на 2
равно %a", x, iquo(x,2)); x:=iquo(x,2); end do;
Остаток от деления 19 на 2 равен 1
```

$$s := 1$$

```
Частное от деления 19 на 2 равно 9
```

$$x := 9$$

```
Остаток от деления 9 на 2 равен 1
```

$$s := 1, 1$$

```
Частное от деления 9 на 2 равно 4
```

$$x := 4$$

```
Остаток от деления 4 на 2 равен 0
```

$$s := 1, 1, 0$$

```
Частное от деления 4 на 2 равно 2
```

$$x := 2$$

```
Остаток от деления 2 на 2 равен 0
```

$$s := 1, 1, 0, 0$$

```
Частное от деления 2 на 2 равно 1
```

$$x := 1$$

```
Остаток от деления 1 на 2 равен 1
```

$$s := 1, 1, 0, 0, 1$$

```
Частное от деления 1 на 2 равно 0
```

$$x := 0$$

```
> convert(19, base, 2);
```

$$[1, 1, 0, 0, 1]$$

```
> convert(19,'binary');
```

$$10011$$

Цикл for/from

for *counter* **from** *initial* **by** *increment* **to** *final* **do** *expressions* **end do;**

Начальное и *конечное значение* счетчика цикла, а также *шаг* счетчика цикла, должны быть числами или переменными с числовыми значениями.

Значения по умолчанию:

- для начального значения счетчика цикла: *initial*=1

- для шага счетчика цикла: *increment*=1
- для конечного значения счетчика цикла: *final*=∞

МАКСИМАЛЬНО СОКРАЩЕННАЯ ФОРМА: **for** *counter* **to** *final* **do** *expressions* **end do;**

Пример

```
> for j from 4 to 8 do print(j) end do
```

4

5

6

7

8

Пример. Четные числа от 8 до 2 в убывающем порядке

```
> for j from 8 to 2 by -1 do if irem(j, 2) = 0 then print(j) end if end do
```

8

6

4

2

```
> j;
```

1

Пример. Вычисление суммы

```
> x := 3.5 : n := 6 :
> s := 0 :
> for j from 1 to n do s := s + j·x end do;
```

$s := 3.5$

$s := 10.5$

$s := 21.0$

$s := 35.0$

$s := 52.5$

$s := 73.5$

```
> s;
```

73.5

Цикл for/in

for *variable* **in** *expression* **do** *expressions* **end do;**

Цикл с параметром for/in работает следующим образом. Параметр последовательно принимает значение элементов выражения Maple (например, слагаемых в сумме, сомножителей в произведении, символов в строке, элементов списка, множества и т. д.) Таким образом, начальным значением параметра является первый элемент выражения Maple, а конечным значением – последний элемент. Как только будут исчерпаны все элементы выражения Maple, произойдет выход из структуры цикла.

Пример 1.

> $f := x^2 + 3x + \frac{1}{x}$:

> **for** s **in** f **do** s; **end do**;

$$x^2$$

$$3x$$

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

Пример 2.

> s := "цикл" :

> **for** i **in** s **do** print(i) **end do**

"ц"

"и"

"к"

"л"

Пример 3. Вычисление суммы тех чисел в диапазоне от 1 до 100, которые являются полными квадратами.

> *restart*;

> $SQR := \{i^2 \$ i = 1..10\}$;

$$SQR := \{1, 4, 9, 16, 25, 36, 49, 64, 81, 100\}$$

> s := 0 :

> **for** i **in** SQR **do** $s := s + i$ **end do**

$s := 1$

$s := 5$

$s := 14$

$s := 30$

$s := 55$

$s := 91$

$s := 140$

$s := 204$

$s := 285$

$s := 385$

Смешанные циклы for/from..while, for/in..while

for *counter* **from** *initial* **by** *increment* **to** *final* **while** *condition* **do** *expressions* **end do;**
for *variable* **in** *expression* **while** *condition* **do** *expressions* **end do;**

Если в полном синтаксисе оператора цикла **for/from** задано конечное значение параметра цикла (конструкция **to** *final*) и присутствует проверка условия (конструкция **while** *condition*), то сначала выполняется сравнение текущего значения параметра цикла с конечным, затем – проверка условия, а после этого выполняется тело цикла.

Если в полном синтаксисе оператора цикла **for/in** присутствует проверка условия (конструкция **while** *condition*), то для каждого значения параметра сначала выполняется проверка условия, а после этого выполняется тело цикла. Параметр последовательно принимает значение элементов выражения Maple до тех пор, пока не встретится элемент,

для которого условие станет ложным. При этом выход из структуры цикла может произойти раньше, чем будут исчерпаны все элементы выражения Maple (см. пример ниже).

Пример 1.

```
> for x from 1 by 2 while x < 6 do print(x) end do;
```

1

3

5

Пример 2. Вывод на экран тех элементов списка, которые являются числами.
Выход из цикла происходит уже на втором элементе списка.

```
> restart;
> s := [2, a, b, 3.5, c]
```

$$s := [2, a, b, 3.5, c]$$

```
> for x in s while type(x, numeric) do print(x) end do
```

2

Пример 3. Вывод на экран последовательности первых десяти простых чисел.

```
> COUNT:=0: PRIMES:=NULL:
```
(сначала задается пустая последовательность)

```
> for n from 1 while COUNT<10 do
if isprime(n) then COUNT:=COUNT+1; PRIMES:=PRIMES,n end if
end do;
```

Теперь выведем полученную в результате работы цикла последовательность простых чисел:

```
> PRIMES;
```

2, 3, 5, 7, 11, 13, 17, 19, 23, 29

Также выведем значение переменной COUNT, чтобы убедиться, что найдено 10 чисел:

```
> COUNT;
```

10

§3. Итеративные команды Maple

Оператор формирования последовательности выражений

expr **$** *name* = *initial* **..** *final***;**

Примеры

```
> $ 2..5;
```

2, 3, 4, 5

```
> a[i] $ i = 1..3;
```

a_1, a_2, a_3

Создание последовательности

seq(*expression***,** *name* = *initial* **..** *final*,*step***);** #аналог цикла **for/from**

Примеры

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

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

```
> seq( i, i="a".."f" );
```

"a", "b", "c", "d", "e", "f"

```
> seq( x[i], i=1..5 );
```

$$x_1, x_2, x_3, x_4, x_5$$

seq(*expression***,** *name* **in** *expression***);** #аналог цикла **for/in**

Примеры

```
> seq(u, u in [Pi/4, Pi^2/2, 1/Pi]);
```

$$\frac{\pi}{4}, \frac{\pi^2}{2}, \frac{1}{\pi}$$

Вычисление суммы(начальные и конечные значения должны быть числами)

add(*expression***,** *name* = *initial* **..** *final***);** #аналог цикла **for/from** **add(***expression***,** *name* **in** *expression***);** #аналог цикла **for/in**

Примеры

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

$$55$$

```
> add(u, u in [Pi/4, Pi/2, Pi/6]);
```

$$\frac{11}{12}\pi$$

Вычисление произведения(начальные и конечные значения должны быть числами)

mul(*expression***,** *name* = *initial* **..** *final***);** #аналог цикла **for/from** **mul(***expression***,** *name* **in** *expression***);** #аналог цикла **for/in**

Примеры

```
> mul(i, i=1..5 );
```

$$120$$

```
> mul(u, u in [Pi/4, Pi^2/2, 1/Pi]);
```

$$\frac{1}{8}\pi^2$$

Команды извлечения и удаления

select(*proc*,*expression***);** #аналог цикла **for/in**
remove(*proc*,*expression***);**
selectremove(*proc*,*expression***);**

select – извлекает те операнды из выражения *expression*, которые удовлетворяют булевой функции (процедуре) *proc* (т.е. оставляет те операнды, значение функции *proc* для которых истинно). Полученный объект имеет тот же тип, что и объект *expression.*

remove – извлекает те операнды из выражения *expression*, которые не удовлетворяют булевой функции (процедуре) *proc* (т.е. оставляет те операнды, значение функции *proc* для которых ложно). Полученный объект имеет тот же тип, что и объект *expression.*

selectremove – сначала извлекает те операнды из выражения *expression*, которые удовлетворяют булевой функции (процедуре) *proc*, а затем те, которые ей не удовлетворяют. Полученные объекты имеют тот же тип, что и объект *expression.*

Примеры

```
> integers := [$10..20];
```

$$integers := [10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20]$$

Выберем из списка **integers** все простые числа. Полученный объект будет списком.

```
> select(isprime, integers);
```

$$[11, 13, 17, 19]$$

Исключим списка **integers** все простые числа. Полученный объект будет списком.

```
> remove(isprime, integers);
```

$$[10, 12, 14, 15, 16, 18, 20]$$

Теперь сформируем два списка

```
> selectremove(isprime, integers);
```

$$[11, 13, 17, 19], [10, 12, 14, 15, 16, 18, 20]$$

Применение команды или процедуры к каждому члену выражения

map(*proc*,*expression***);** #аналог цикла **for/in**

Примеры

```
> map(f, {a,b,c});
```

$$\{f(a), f(b), f(c)\}$$

```
> map(f, x + y*z);
```

$$f(x) + f(y\,z)$$

```
> map(f, y*z);
```

$$f(y)\,f(z)$$

```
> map(sqrt,a+b);
```

$$\sqrt{a} + \sqrt{b}$$

```
> f := x -> sin(x + 3) :
> map(f, y·z);
```

$$\sin(y + 3)\,\sin(z + 3)$$

Разложим на множители все составные числа из списка **integers**.

```
> integers := [$10..20]: comp:=remove(isprime,integers);
map(ifactor,comp);
```

$$comp := [10, 12, 14, 15, 16, 18, 20]$$

$$[(2)\,(5), (2)^2\,(3), (2)\,(7), (3)\,(5), (2)^4,$$
$$(2)\,(3)^2, (2)^2\,(5)]$$