Тема 5. Оператори умови та циклу. Послідовності

Оператори це команди для роботи комп'ютера над програмою.

Оператори представляються відповідними **ідентифікаторами** у вигляді спеціальних математичних знаків та слів.

Оператор умови *if*

Основу будь-якого обчислювального пристрою складають елементарні логічні схеми, які базуються на законах **алгебри логіки.**

Апарат алгебри логіки було розроблено в 1854 р. Дж. Булем, як спробу вивчити логіку мислення математичними методами.

Основними логічними операціями є:

- *and* – логічне множення або ж кон'юнкція;
- *or* – логічне додавання або ж диз'юнкція;
- *not* – логічне віднімання або ж інверсія.

Ці логічні операції дозволяють сформувати складні логічні вирази.

Таблиці істинності:

a	*and*	*b*
1	*1*	*1*
1	*0*	*0*
0	*0*	*1*
0	*0*	*0*

a	*or*	*b*
1	*1*	*1*
1	*1*	*0*
0	*1*	*1*
0	*0*	*0*

Операції відношення

Операції відношення <, >, <=, >=, = і <> зв'язують два алгебраїчних вирази, формують новий тип вираз.

В *Maple* розгалуження реалізується оператором *if*, загальна форма якого має синтаксис:

if булевий_вираз then послідовність_операторів
[Elif булевий_вираз then послідовність_операторів]
[Else послідовність_операторів]
end if

Якщо *булевий вираз* є істинним після ключового слова *if*, то виконується послідовність операторів після ключового слова *then* до першого *elif, else* або *end if.*

Якщо значення *булевий вираз* дорівнює *false*, то перевіряється на істинність *булевий вираз* після ключового слова *elif*, якщо воно задано, і в разі істинності виконуються оператори після ключового *then*.

Якщо жодне з булевих умов неправдиве, то виконуються оператори блоку *else*, знову-таки, в випадку його завдання.

Блоків *elif* може бути скільки завгодно, тоді як блок *else* завжди тільки один.

Приклади.

```
x:=1: g:=0:
if x <0 then g:=- 1; end if;
if x <0 then g:=- 1; else g:=1; end if;
if x <0 then g:=- 1;
        elif x <1 then g:=0; else g:=1; end if;
```

Оператор `if`

Оператор `if` має синтаксис: *`if` (умова, операнд1, операнд2)*.

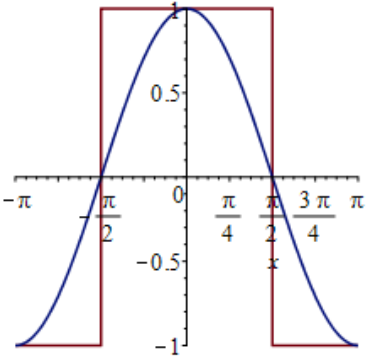
Семантика оператора `if`: спочатку обчислюється значення умови, яке повинно бути булевим виразом, і якщо воно *true*, то результатом операції буде *операнд1*, в іншому випадку *операнд2*.

```
a:=4: b:=3:
c:=`if`(a>b, a, b)+sin(`if`(a>b, a, b));
```

$$c := 4 + \sin(4)$$

Приклад.

```
plot([`if`(cos(x)>0, 1, -1), cos(x)], x=-Pi..Pi);
```



У булевих відносинах в операторі *if* відношення може бути рівним *true* або *false*. Обчислити відношення в булевому контексті можна командою *evalb()*.

Оператори циклу

Для організації повторюваних обчислень в *Maple* передбачені дві форми операторів циклу: *for-from* і *for-in*.

Перший оператор циклу є універсальним і включає в себе як цикли, повторювані задане число разів, так і цикли, що виконуються поки деякий булевий вираз є істинним.

Друга форма циклу *for* реалізує цикл по елементах списку або множини, і в інших мовах програмування він відомий як цикл *for-each*.

Оператор циклу *for-from*

Загальний синтаксис оператора циклу *for-from* наступний:

[for ім'я] [from вираз] [by вираз] [to вираз] [while булевий_вираз]
do expr end do;

або ж:

for var from val1 by val2 while bool1 do expr end do;

Тіло циклу *expr* виконується, поки логічне вираження *bool1* є істинним, змінна *var* змінюється від значення *val1* з кроком *val2*. Наприклад:

for x from 1 by 2 while x <6 do print (x) od;

1

У блоці *for* задається ім’я змінної циклу, блоки *from* та *to* визначають, відповідно, початкове і кінцеве значення діапазону зміни змінної циклу, а в блоці *by* задається крок її зміни (він може бути і негативним).

Виконання циклу починається з присвоювання змінної циклу початкового значення, після чого перевіряється, чи не перевершує вона кінцевого значення, і в разі позитивного відповіді, виконуються оператори тіла циклу, задані в блоці *do … end do*, змінна циклу збільшується на значення кроку і алгоритм перевірки починається знову.

Якщо значення змінної циклу перевершує кінцеве значення, то цикл припиняє своє виконання. Або якщо при перевірці початкового

значення змінної циклу, воно перевершує кінцеве значення, то цикл завершує своє виконання, а оператори тіла циклу не виконуються жодного разу.

Якщо заданий блок *while*, то одночасно з перевіркою значення змінної циклу перевіряється на істинність булевого виразу цього блоку, і цикл також завершує роботу, якщо його значення виявляється рівним *false*.

Всі перераховані блоки є необов'язковими і можуть задаватися в довільному порядку за одним винятком: якщо присутній блок *for*, то він повинен бути заданий першим.

Якщо який-небудь блок не заданий, то його параметр приймає значення за замовчуванням:

Приклад.

for i from 1 to 10 do evalf (sqrt(i)) end do;

Оператор циклу *for-in*

Друга форма оператора циклу *for-in* організовує цикл по елементах об’єкта, який може бути представлений послідовністю, списком, сумою, добутком або рядком.

Загальний синтаксис оператора циклу *for-in* має вигляд:

for ім’я in об’єкт [while булевий_вираз] do
послідовність_операторів
end do;

Змінна циклу, що визначається в блоці *for … in*, послідовно приймає значення операндів *об’єкта*.

Цикл виконується стільки разів, скільки операндів задано в об’єкті.

Приклад. Підсумовування елементів множини

S:={x, y, z}: s:=0:
for I in S do s:=s+I end do: s;

x+y+z

Якщо необхідно перервати подальше виконання ітерацій циклу при виконанні деякої умови, то потрібно використати оператор *break*. Він використовується спільно з оператором умови *if*.

Приклад. Підсумовування перших двох елементів множини

S:={x, y, z}: s:=0: j:=1:
for I in S do s:=s+I; if j>2 then break; end if; j:=j+2; end do: s; j;

$$\frac{x+y}{3}$$

Цикл можна перервати за допомогою блока *while*.

for I from 1 to 5 while i<6 do print(i) end do;

Цикл в залежності заданого списку:

for i in [1,2,3,4,5] do print(i) end do;

Оператор циклу *while*

Оператор циклу типу *while (‘поки’)* має вигляд:

while bool do expr od.

Тіло циклу *expr* виконується, поки значення логічного виразу *bool=true*, і виконання припиняється, якщо *bool=false*.

Наприклад:

```
j:=0:
while j <5 do j:=(j+1)^j od;
```

Оператори створення послідовностей

Створення елементів списку:

```
L:=[seq(i, i=1..10)];
```

L:=[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

Вибір непарних

```
select(type, L, ‘odd’);
```

[1, 3, 5, 7, 9]

Видалення непарних

```
remove(type, L, ‘odd’);
```

[2, 4, 6, 8, 10]