SCILAB

Глава 3. Программирование

Содержание главы:

- Использование функций
- Структура функций
- Загрузка функций
- Глобальные и локальные переменные
- Специальные команды для функций
- Определение операций на новых типах данных
- Отладка (debbuging) Scilab-функции

Использование функций

В пакете есть возможность использовать функции. Это позволяет создавать интегрированные в Scilab специализированные программы и использовать библиотеки. Средства программирования (Tools): циклы, условные конструкции. Разрешено применение вложенных циклов.

В условных конструкциях и циклах используются следующие операторы сравнения:
== равно
< меньше чем
> больше чем
<= меньше или равно чем
>= больше или равно чем
<> или ~= не равно

Определены следующие типы циклов:
for
while

1) Цикл **for**
Синтаксис
for variable=n1:step:n2,<тело цикла>,end
В качестве индекса цикла могут выступать переменные различных типов (вектора, списки и др.).
Замечание:
Если шаг цикла `step=1`, его можно не писать: `x=1;for k=1:4,x=x*k,end` эквивалентно `x=1;for k=1:1:4,x=x*k,end`.

Пример цикла **for**.
Вычисление факториала

```
-->x=1;for k=1:4,x=x*k,end
x =
1.
x =
2.
x =
6.
x =
24.
```

Замечание: Поскольку внутри цикла в этом примере стоят в качестве разделителя стоят запятые (","), выводятся все промежуточные значения "x". Если использовать конструкцию с разделителем "точка с запятой" (";"), то никаких сообщений ни будет и по специальному запросу можно будет узнать конечное значение "x".

```
x=1;for k=1:4;x=x*k;end
```

2)Цикл **while**
Синтаксис
while <тело цикла> end
Пример цикла **while**.

```
-->x=1; while x<14, x=x*2,end
x =
2.
x =
4.
x =
8.
x =
16.
```

Циклы **while** и **for** могут быть прерваны с помощью оператора **break**.
Пример.

```
-->a=0; for i=1:5:100, a=a+1; if i>10 then break, end;end
-->a
a =
3.
```

Замечание: Если мы хотим прервать извне выполнение цикла (например, видно, что произошло зацикливание), используйте функционального меню управления окна: **Control - Abort**. Посмотрите также команды **return** и **pause**.

Допустимы следующие условные конструкции:
1) **if-then-else**
2) **select-case**

1) Конструкция **if-then-else**
Синтаксис
if expr1 then statements
elseif expri then statements
....
else statements
end,
end
Замечание: Число символьных знаков, используемых для определения тела условной конструкции (**if while for** или **select/case**) должно быть ограничено 16k.

Пример.
```
i=2
for j = 1:3,
if i == j then
a(i,j) = 2;
elseif abs(i-j) == 1 then
a(i,j) = -1;
else a(i,j) = 0;
end,
end
```
Результат:
```
a =
!   0.    0.    0. !
! - 1.    2. - 1. !
```
Там, где, например, в языке C++ используются фигурные скобки **{ }** для разделения текста, в Scilab используется запятая.

Синтаксис конструкции " **select-case**"
```
select expr0,
case expr1 then instructions1,
case expr2 then instructions2,
...
case exprn then instructionsn,
[else instructions],
end
```
Пример.
```
x=-1
select x, case 1,y=x+5,case -1,y=sqrt(36),end
```
Результат:
```
y=6
```

Структура функций

Функции играют роль подпрограмм. Удобнее всего набирать функции в текстовом редакторе и хранить их в отдельных файлах (внешние функции), но можно их использовать и непосредственно в системе Scilab.

Синтаксис
function[y1,...,yn]=foo(x1,...,xm)
..... тело функции ....
endfunction
Где
foo - имя фунции,
xi - входные аргументы функции (их **m** штук),
yi - выходные аргументы функции (их **n** штук).

Пример.
Вычисление факториала.
```
function [x]=fact(k)
k=int(k)
if k<1 then k=1, end
x=1;
for j=1:k,x=x*j;end
endfunction
```

Наберем этот текст в любом текстовом редакторе и сохраним его в файле с именем `fact.sci`. Расширение ***.sci** является для Sclab "родным", но не обязательным. Затем следует вызвать эти файлы с из Scilab помощью команд **getf(filename)** или **exec(filename,-1);** Те же операции можно произвести с помощью команд меню **File-getf** или **File-exec**.
Для нашего примера этот вызов будет:

```
getf('D:\zzz\fact.sci');
```

До вызова функции желательно проверить, не была ли уже загружена такая функция ранее. Для этого:

```
exists('fact')
```

Результат:

```
ans =
0.
```

Так же можно было использовать для этого команду **who**.
После загрузки функции с помощью

```
-->exec('D:\zzz\fact.sci');
-->exists('fact')
```

получим:

```
ans =
1.
```

Теперь мы можем пользоваться этой функцией:

```
-->x=fact(5)
x =
120
```

Загрузка функций

Для загрузки функций применяются команды **exec**, **deff** и **getf**.
При определении функции **function[y1,...,yn]=foo(x1,...,xm)** входные и выходные параметры **xi** и **yi** могут быть любыми объектами Scilab, кроме самих этих функций. Если функция предварительно не загружена в Scilab с помощью **getf(filename)** или **exec(filename,-1)**, то она и не выполняется. Загружаемый файл может содержать несколько функций. Функции можно определять и непосредственно в сеансе Scilab, используя конструкцию **function/endfunction** или используя функцию **deff**. Это полезно, если мы хотим определить функцию как выходной параметр другой функции.

Способ 1.
Вызов функции с помощью **exec**.
Синтаксис
exec(fun [,mode])
exec(path [,mode])
ierr=exec(path,'errcatch' [,mode])
ierr=exec(fun,'errcatch' [,mode])
Параметры
fun : имя файла, в котором находится Scilab функция (функция определена непосредственно в Scilab)
path : строка, содержащая полный путь к файлу, содержащему функцию
ierr : целое, 0 или номер ошибки
mode : целый скаляр, могущий принимать следующие выражения:
0 : по умолчанию
-1 : ничего не печатать
1 : сообщение (echo) для каждой командной строки
2 : prompt --> печатает
3 : echoes + prompts

4 : stops остановка перед каждым "prompt"
7 : stops + prompts + echoes : полезно для демонстраций.

Пример.
```
exec('C:\scilab\macros\algebre\aff2ab.sci')
```

Замечание: Файл `aff2ab.sci` является файлом сценария. Подробно об использовании файлов сценария смотрите раздел "Как использовать файл сценария?" в главе 2.

Способ 2.
Вызов функции с помощью **deff**
Синтаксис
deff('[s1,s2,...]=newfunction(e1,e2,....)',text [,opt])
Параметры
e1,e2,..., : входные переменные
s1,s2,..., : выходные переменные
text : матрица из символьных значений (character strings)
opt : необязательная переменная типа character string. Возможные значения **opt**:
'c' : функция скомпилирована для большей эффективности (по умолчанию)
'n' : функция не скомпилирована

Пример.
```
deff('[x]=myplus(y,z)','x=y+z')
deff('[x]=mymacro(y,z)',['a=3*y+1'; 'x=a*z+y'])
```
Замечание: на первый взгляд это выглядит достаточно неудобно.

Способ 3.
Вызов функции с помощью **getf**
Синтаксис
getf(file-name [,opt])
Параметры
filename : Имя файла (Scilab string)
opt : необязательная переменная типа character string. Возможные значения **opt**:
'c' : функция скомпилированна для большей эффективности (по умолчанию)
'n' : функция нескомпилирована
"p" : загружаемая функция скомпилирована и приготовлена для профилирования.

Пример.
```
getf('SCI/macros/xdess/plot.sci')
```
Замечание: Метод вызова функций с помощью **getf** считается устаревшим, предпочтительнее пользоваться **exec**.

Глобальные и локальные переменные

Переменные могут являться локальными и глобальными.

1) **Локальные переменные**
Если значения переменных в функции не определены (и не являются входными параметрами), то их значения берутся как переменные из вызывающей среды.

Пример.
Наберем в любом текстовом редакторе в ASCII кодах:

```
function [y1,y2]=f(x1,x2)
y1=x1+x2;
y2=x1-x2;
endfunction
```

Запишем эту функцию в файл `D:\zzz\fact.sci`.

```
exec('D:\zzz\fact.sci') // загрузка функции
[y1,y2]=f(1,1)
```

Результат:

```
y2 =
0.
y1 =
2.
```

Лучше все же при вызове пользоваться не именами переменных, используемых в определении функции, а вводить новые.
Пример.

```
[a,b]=f(1,1)
```

При этом x1 x2 остаются локальными переменными и после вызова функции остаются неизвестными для Scilab.

2)**Глобальные переменные**
Могут быть определены с помощью команды **global**.
Синтаксис
global('nam1',...,'namn')
global nam1 ... namn
Параметры
nam1,..., namn : имена переменных
Пример.

```
global a b c
a=1;b=2;c=3;
```

Для уничтожения глобальных переменных служит команда **clearglobal**.
clearglobal() уничтожает все глобальные переменные.
clearglobal nam1 .. namn уничтожает глобальные переменные с заданными именами.
clearglobal nam1 .. namn
clearglobal('nam1', ..,'namn')
Чтобы узнать, какие глобальные переменные определены в данной сессии, служит команда **who('global')**.
Пример.

```
who('global')
```

Посмотрите, какие из глобальных переменных остались с помощью `who('global')`.
clearglobal('a') `clearglobal()` уничтожит все переменные и результатом `who('global')` будет:

```
ans = []
```

Специальные команды для функций

argn	возвращает число входных и выходных аргументов в вызове функции
error	используется для печати сообщений об ошибке или разветвлений на случай детектирования ошибок.
pause	мода ожидания. Используется для отладки.

abort	служит для выхода из режима ожидания
warning	ожидание сообщений
break	прерывание цикла
return и **resume**	служат для передачи локальных переменных из функции в основное тело программы

Замечание: Все эти служебные команды используются исключительно внутри функций.
Пример.

```
function [z]=foo(x,y)
if x==0 then,
error('division by zero');
end,
z=x/y
endfunction
```

Результат применения функции foo:

```
-->foo(6,2)
ans =
3.
-->foo(6,0)
!--error 27
division by zero...
at line 5 of function foo called by :
foo(6,0)
```

Определение операций на новых типах данных

Существует возможность определить фундаментальные операции для новых типов данных. Смотри **help overloading**. Пользователь может дать собственное определение, например, операции умножения. Новые операторы, определенные пользователем, должны начинаться со знака **%** и состоять из 3 или 4 полей. Смотрите таблицу для определения возможных имен на стр. 68 файла **intro.pdf** или с помощью **help overloading**. Оставим рассмотрение этой возможности на будущее.

Отладка (debbuging) Scilab-функции

Наиболее простой способ отладки Scilab-функции состоит во введении в функцию одной из специальных команд (смотри главу 3.2.4). Остановить выполнение функции можно с помощью оператора **abort**.Удобно вставлять в функцию прерывания. Посмотрите команды **setbpt**, **delbpt** и **disbpt**. Для отслеживания ошибок также удобны **errclear** и **errcatch**. Эксперты в Scilab могут использовать функцию **debug(i)**, где i=0,...,4) означают уровень отладки.