Компьютерные технологии в научных исследованиях и образовании

Юрий Анатольевич Богоявленский, заведующий кафедрой Информатики и математического обеспечения, к.т.н., доцент, ybgv

Функции

Определение функций

Контроль фактических параметров функции

Несколько возвращаемых значений

Файлы с текстами функций

Управление путем загрузки файлов функций

Подфункции

Глобальные и локальные переменные

Постоянные (`persistent`) переменные

Фиксация функций в памяти

Приоритет функций

Функцию можно определить непосредственно в командной строке во время интерактивных сеансов Octave или во внешнем файле. При этом она может быть вызвана для выполнения так же, как встроенная функция.

Файлы содержащие функции или скрипты должны иметь расширение `.m` для совместимости с `matlab`. Если нужно, чтобы в файле было более одной независимой функции, это должен быть файл сценария, который необходимо выполнить перед использование этих функций.

Определение функций

Определение функции имеет вид:

```
function ret-var = name (arg-list)
```

```
  тело
  return
endfunction
```

где:

- тело — содержит последовательность инструкций octave, определяющих алгоритм работы функции;
- ret-var — имя переменной любого допустимого типа, которой будет присвоено значение результата выполнения функции. В теле должна быть по крайней мере одно выражение присваивания значения этой переменной;
- name — имя функции которое конструируется по тем же правилам, что и имя переменной;
- (arg-list) — разделенный запятыми список формальных параметров функции;
- return — обязательная инструкция возврата управления в вызывающую функцию.

При вызове формальные параметры заменяются на фактичские и происходит передача управления на инструкции тела функции, поэтому при любом завершении их выполнения необходимо чтобы последней выполненной телом инструкцией была инструкция return.

Отметим, что ret-var и (arg-list) может быть пустыми. Если пусты оба, то функция задается в виде:

```
function name
  тело
  return
endfunction
```

Эта форма задания функции не имеет средств для задания исходных данных и получения результатов, ее можно применять для определения, например, каких-нибудь системных действий.

Если пуст `ret-var,` то функция задается в виде:

```
function name (arg-list)
  тело
  return
endfunction
```

Пример.

```
ybgv@ybgv-home:~> octave -q -p ~/MyOct
octave:1>
octave:1> newwhos
octave:2>
octave:2> function Hello (message)
> printf("%s\n", message)
> return
> endfunction
octave:3>
octave:3> Hello("Hei")
Hei
octave:4>
octave:4> Hello("Эфес о ладонь согреешь!")
Эфес о ладонь согреешь!
octave:5>
```

Контроль фактических параметров функции

Переменные, используемые в теле функции, являются локальными для функции. Переменные, названные

в arg-list и ret-var также являются локальными для функции. Смотрите Раздел 7.1 [Глобальные переменные],

стр. 139, для получения информации о том, как получить доступ к глобальным переменным внутри функции.

Такой контроль облегчает отладку и повышает производительность труда и особенно актуален для языка `octave`, который является слабо типизированным, что, в частности означает наличие возможности переменной изменить тип.

Этот контроль реализуется проверкой необходимых свойств фактических параметров в первых инструкциях тела определяемой функции. При этом используются следующие встроенные функции `octave.`

Предотвратить задание фактического параметра неверного типа можно с помощью функций определения типа переменной, таких как, напрмер, `isnumeric (x), isvector (x)` и др. (см. полный список в [Док, с. 67 — 70]).

Предотвратить задание неверного количества фактических параметров можно с помощью функций `nargin (), nargin (fcn)`, которые выдают количество параметров, заданных при определении функции (см. детали в [Док, с. 193 — 194]).

Для контроля параметров также можно использовать не рассматриваемую нами функцию `inputname (n)`, описанную в [Док, с. 67 — 70]. Например, вот функция, которая вычисляет среднее значение элементов вектора:

```
ybgv@ybgv-home:~> octave -q p ~/MyOct
octave:1>
octave:1> newwhos
```

```
octave:2>
octave:2> unix ("cd MyOct;less avg.m")
function retval = avg (v)
    retval = 0;
    if (nargin != 1)
        error ("avg: параметр не один, вызов: avg (vector)");
    endif

    if (isvector (v))
        retval = sum (v) / length (v);

    else
    error ("avg: параметр не вектор");

    endif
    return
endfunction
ans = 0
octave:3> x=[2.3 5.6 7.25 100.3 9.5 10]
x =

    2.3000    5.6000    7.2500  100.3000    9.5000   10.0000

octave:4> y=[3 4;5 6]
y =

   3   4
   5   6

octave:5> z=[1+2i 3 5.5 6.7 6+2i]
z =
```

```
 Columns 1 through 3:

   1.0000 + 2.0000i   3.0000 +      0i   5.5000 +      0i

 Columns 4 and 5:

   6.7000 +      0i   6.0000 + 2.0000i

octave:6> avg
error: avg: параметр не один, вызов: avg (vector)
error: called from
    avg at line 4 column 2
octave:7> avg(x)
ans = 22.492
octave:8> avg(y)
error: avg: параметр не вектор
error: called from
    avg at line 11 column 5
octave:9> avg(z)
ans =  4.4400 + 0.8000i
octave:10> whos
Variables visible from the current scope:

variables in scope: top scope

Attr   Name        Size           Type                Bytes Class
====   ====        ====           ====                ===== =====
c      ans         1x1            complex scalar         16 double
       x           1x6            matrix                 48 double
       y           2x2            matrix                 32 double
c      z           1x5            complex matrix         80 double

Total is 16 elements using 176 bytes
```

```
octave:11> w=[2;3;4;5;6]
w =

   2
   3
   4
   5
   6

octave:12> avg(w)
ans = 4
octave:13> whos
Variables visible from the current scope:

variables in scope: top scope

Attr   Name        Size            Type                  Bytes Class
====   ====        ====            ====                  ===== =====
       ans         1x1             scalar                    8 double
       w           5x1             matrix                   40 double
       x           1x6             matrix                   48 double
       y           2x2             matrix                   32 double
c      z           1x5             complex matrix           80 double

Total is 21 elements using 208 bytes

octave:14>
```

Отметим, что в [Док, с. 202 — 210] представлен широкий набор функций octave для проверки различных свойств фактических параметров.

Несколько возвращаемых значений

Если результатом выполнения функции является несколько значений, то функцию можно определить в виде:

```
function [ret-list] = name (arg-list)
  тело
  return
endfunction
```

где `[ret-list]` — разделенный запятыми список имен переменных, которые будут содержать значения, возвращаемые из функции, он должен содержать хотя бы один элемент. Рассмотрим пример функции, которая возвращает два значения: максимальный элемент вектора и индекс его первого появления в векторе. Отметим, что при вызове такой функции вида `name (arg-list)` встроенная переменная ans будет иметь значение полученного функцией первого элемента списка `[ret-list]`.

```
ybgv@ybgv-home:~> octave -q -p ~/MyOct
octave:1>
octave:1> newwhos
octave:2> x=[0 1 2 3 2 1 0 3 2 1 3]
x =


   0   1   2   3   2   1   0   3   2   1   3


octave:3> unix("cd MyOct;less vmax.m")
function [max, idx] = vmax (v)
    idx = 1;
    max = v (idx);
    for i = 2:length (v)
        if (v (i) > max)
```

```
            max = v (i);
            idx = i;
        endif
    endfor
    return
endfunction
ans = 0
octave:4> [a,b]=vmax(x)
a = 3
b = 4
octave:5> vmax(x)
ans = 3
octave:6> whos
Variables visible from the current scope:

variables in scope: top scope

Attr   Name        Size            Type          Bytes Class
====   ====        ====            ====          ===== =====
       a           1x1             scalar            8 double
       ans         1x1             scalar            8 double
       b           1x1             scalar            8 double
       x           1x11            matrix           88 double

Total is 14 elements using 112 bytes

octave:7>
```

Функцию nthargout (n, "имя функции", ее arg-list) можно использовать для получения только некоторых возвращаемых значений. Пример

```
octave:9> nthargout (1,"vmax",x)
ans = 3
octave:10> nthargout (2,"vmax",x)
ans = 4
octave:11>
```

Другие, не рассматриваемые нами, возможности этой функции, а также функции `nargout`, представлены в [Док, с. 196 — 197]. Кроме того мы не рассматриваем описанные в [Док, с. 197 — 202] механизмы реализации:

- списка формальных параметров переменной длины;
- списка возвращаемых значений переменной длины;
- фиктивного заполнителя формального параметра;
- значения формального параметра по умолчанию.

Файлы с текстами функций

Тексты функций удобно хранить в файлах. Ручная загрузка этих файлов не требуется, но `octave` нужно задать каталог файловой системы в котром их можно найти. Когда интерпретатор `octave` обнаруживает не определенный ранее идентификатором, он ищет в списке каталогов (`load path`, см. ниже, в разделе «Управление путем загрузки файлов функций») файлы с расширением `.m`, имя которых (без расширения) совпадает с этим идентификатором. Если файл найден, его текст считывается и, если он определяет единственную функцию, она компилируется и выполняется.

При этом интерпретатор `octave` сохраняет полное имя файла время его модификации. Если это время меняется то `octave` может перезагрузить файл. При работе в интерактивном режиме проверка времени обычно происходит каждый раз когда `octave` выводит на экран

приглашение. Поиск новых определений функций также выполняется при изменении текущего рабочего каталога. Такой механизм позволяет редактировать определение функции во время выполнения интерпретатора `octave` и автоматически использовать новое определение функции без необходимости его перезапуска.

Чтобы избежать ненужного снижения производительности путем проверки временных меток для функций, которые вряд ли изменятся, считается, что готовые файлы функций в дереве каталогов home/share/octave/version/m не меняются и их временные метки проверять не надо, что обеспечивает значительное улучшение производительности для файлов функций, которые распространяются вместе с `octave`.

Если вы знаете, что ваши собственные файлы функций не будут изменяться во время выполнения интерпретатора `octave`, то можно повысить производительность отменив проверку временных меток выполнением функци:

```
ignore_function_time_stamp ("all").
```

Проверка меток будет восстановлена выполнением функции:

```
ignore_function_time_stamp ("system").
```

Выполнением функции:

```
ignore_function_time_stamp ("none")
```

Приведет к проверке временных меток, и перекомпиляции при необходимости, всех файлов функций, в том числе тех, которые распространяются вместе с `octave.`

Мы не рассматриваем описанные в [Док, с. 211 — 213] инструменты редактирования файлов и управления их именами.

Управление путем загрузки файлов функций

При вызове функции `octave` выполняет поиск файла, содержащего ее определение в списке каталогов, называемом «`load path`». По умолчанию он содержит список каталогов, распространяемых с помощью Octave, плюс

текущий рабочий каталог. Текущий список можно вывести выполнив функцию path. Для его постраничного просмотра можно воспользоваться функцией more on.

Пример для текущей конфигурации на моей рабочей ЭВМ.

```
bgv@ybgv-home:~> octave -q -p ~/MyOct
octave:1> more on
octave:2> path
Octave's search path contains the following directories:


.
/home/ybgv/MyOct
/usr/lib64/octave/site/oct/api-v57/x86_64-suse-linux-gnu
/usr/lib64/octave/site/oct/x86_64-suse-linux-gnu
/usr/share/octave/7.1.0/site/m
/usr/share/octave/site/api-v57/m
/usr/share/octave/site/m
/usr/share/octave/site/m/startup
/usr/lib64/octave/7.1.0/oct/x86_64-suse-linux-gnu
/usr/share/octave/7.1.0/m
/usr/share/octave/7.1.0/m/audio
/usr/share/octave/7.1.0/m/deprecated
/usr/share/octave/7.1.0/m/elfun
/usr/share/octave/7.1.0/m/general
/usr/share/octave/7.1.0/m/geometry
/usr/share/octave/7.1.0/m/gui
/usr/share/octave/7.1.0/m/help
/usr/share/octave/7.1.0/m/image
/usr/share/octave/7.1.0/m/io
/usr/share/octave/7.1.0/m/java
/usr/share/octave/7.1.0/m/legacy
/usr/share/octave/7.1.0/m/linear-algebra
```

```
/usr/share/octave/7.1.0/m/miscellaneous
/usr/share/octave/7.1.0/m/ode
/usr/share/octave/7.1.0/m/optimization
/usr/share/octave/7.1.0/m/path
lines 1-29
/usr/share/octave/7.1.0/m/io
/usr/share/octave/7.1.0/m/java
/usr/share/octave/7.1.0/m/legacy
/usr/share/octave/7.1.0/m/linear-algebra
/usr/share/octave/7.1.0/m/miscellaneous
/usr/share/octave/7.1.0/m/ode
/usr/share/octave/7.1.0/m/optimization
/usr/share/octave/7.1.0/m/path
/usr/share/octave/7.1.0/m/pkg
/usr/share/octave/7.1.0/m/plot
/usr/share/octave/7.1.0/m/plot/appearance
/usr/share/octave/7.1.0/m/plot/draw
/usr/share/octave/7.1.0/m/plot/util
/usr/share/octave/7.1.0/m/polynomial
/usr/share/octave/7.1.0/m/prefs
/usr/share/octave/7.1.0/m/profiler
/usr/share/octave/7.1.0/m/set
/usr/share/octave/7.1.0/m/signal
/usr/share/octave/7.1.0/m/sparse
/usr/share/octave/7.1.0/m/specfun
/usr/share/octave/7.1.0/m/special-matrix
/usr/share/octave/7.1.0/m/startup
/usr/share/octave/7.1.0/m/statistics
/usr/share/octave/7.1.0/m/strings
/usr/share/octave/7.1.0/m/testfun
/usr/share/octave/7.1.0/m/time
/usr/share/octave/7.1.0/m/web
```

```
  /usr/share/octave/7.1.0/data
  /usr/share/octave/7.1.0/data

  lines 22-50/50 (END)
```

В octave реализован полноценный набор функций, необходимых для управления load path. Так вызов функции addpath ("~/Some-Dir") добавит в список каталог ~/Some-Dir, а вызов функции rmpath ("~/Some-Dir") удалит его из списка. Варианты этих функций и другие функции управление load path даны в [Док, с. 213 — 215]).

Доступ к функции через дескриптор

Вызов функции можно обобщить с помощью ее дескриптора получаемого с помощью специального оператора ‘@’. Дескриторы можно и использовать там, где неудобно использовать имя функции, например пр работе с рисунками и графикой. Пример.

```
ybgv@ybgv-home:~> octave -q -p ~/MyOct
octave:1>
octave:1> newwhos
octave:2> a=[1 2 3 4 5 6 7 8 9 10]
a =

    1    2    3    4    5    6    7    8    9   10

octave:3> x=@plus
x = @plus
octave:4> y=@avg
y = @avg
```

```
octave:5> x(3,5)
ans = 8
octave:6> plus(3,5)
ans = 8
octave:7> y(a)
ans = 5.5000
octave:8> avg(a)
ans = 5.5000
octave:9> whos
Variables visible from the current scope:

variables in scope: top scope

Attr   Name        Size            Type                  Bytes Class
====   ====        ====            ====                  ===== =====
       a           1x10            matrix                   80 double
       ans         1x1             scalar                    8 double
       x         1x1             function handle             0 function_handle
       y         1x1             function handle             0 function_handle

Total is 13 elements using 88 bytes

octave:10>
```

Подфункции

Файл функции может содержать определение других функций, называемых подфункциями. Они доступны для вызова только функциям, определенным в том же самом файле функции. В примере ниже в файле f.m определена основная функция `f` и две подфункции `g` и `h`, которые могут быть вызваны только из основной функции `f` или из других подфункций, но не извне файла `f.m`.

```
ybgv@ybgv-home:~> octave -q -p ~/MyOct
octave:1>
octave:1> unix("cd MyOct;less f.m")

#   Основная функция

function f ()
    localfunctions ()
    printf ("in f, calling g\n");
    g ()
endfunction

#   Подфункция g

function g ()
    printf ("in g, calling h\n");
    h ()
endfunction

#   Подфункция h

function h ()
    printf ("in h\n")
endfunction
ans = 0
octave:2> f
ans =
{
  [1,1] = @g
  [2,1] = @h
}
```

```
in f, calling g
in g, calling h
in h
octave:3> g
error: 'g' undefined near line 1, column 1
octave:4> h
error: 'h' undefined near line 1, column 1
octave:5>
```

Отметим, что функция localfunctions() озвращает список всех подфункций, определенных в текущем файле функций. Возвращаемое значение представляет собой столбцовый массив ячеек содержащий дескриптоы всех подфункций.

Отметим, что мы не рассматриваем частные (private) и вложенные (nested) функции, а также прегрузку и автозагрузку фукций, описанные в [Док, с. 217 — 221].

Глобальные и локальные переменные.

При запуске интерпретатора octave создается исходная область видимости переменных (top scope). При вызове функции создается ее локальная область видимости. В зависимости от области видимости переменные могут быть глобальными, локальными и постоянными (persistent, см. ниже). Рассмотрим простой пример.

```
ybgv@ybgv-home:~> octave -q -p ~/MyOct
octave:1>
octave:1> newwhos
octave:2>
octave:2> global alpha = pi
```

```
octave:3> alpha
alpha = 3.1416
octave:4> x = [1 2 3 5 3 3]
x =

   1   2   3   5   3   3

octave:5> Mx=0,I=0
Mx = 0
I = 0
octave:6> whos
Variables visible from the current scope:

variables in scope: top scope

Attr   Name        Size            Type          Bytes Class
====   ====        ====            ====          ===== =====
       I           1x1             scalar            8 double
       Mx          1x1             scalar            8 double
   g   alpha       1x1             scalar            8 double
       x           1x6             matrix           48 double

Total is 9 elements using 72 bytes

octave:7> unix("cd MyOct;less scvmax.m")
function [max, idx] = scvmax (v)
    global alpha
    idx = 1;
    max = v (idx);
    for i = 2:length (v)
        if (v (i) > max)
            max = v (i);
```

```
            idx = i;
        endif
    endfor
    alpha
    alpha = 2+3i
    whos
    return
endfunction
ans = 0
octave:8> [Mx I] = scvmax(x)
alpha = 3.1416
alpha =  2 + 3i
Variables visible from the current scope:

variables in scope: scvmax: /home/ybgv/MyOct/scvmax.m

Attr   Name        Size            Type                   Bytes Class
====   ====        ====            ====                   ===== =====
c  g   alpha       1x1             complex scalar            16 double
       i           1x1             scalar                     8 double
  f    idx         1x1             scalar                     8 double
  f    max         1x1             scalar                     8 double
  f    v           1x6             matrix                    48 double

Total is 10 elements using 88 bytes

Mx = 5
I = 4
octave:9> alpha
alpha =  2 + 3i
octave:10> whos
Variables visible from the current scope:
```

```
variables in scope: top scope

Attr   Name        Size            Type                    Bytes Class
====   ====        ====            ====                    ===== =====
       I           1x1             scalar                      8 double
       Mx          1x1             scalar                      8 double
c  g   alpha       1x1             complex scalar             16 double
       ans         1x1             scalar                      8 double
       x           1x6             matrix                     48 double

Total is 10 elements using 88 bytes

octave:11>
```

Видны две независимые области видимости top scope и scope: scvmax: /home/ybgv/MyOct/scvmax.m. Переменная alpha, описанная инструкцией global alpha как при запуске интерпретатора в области идимости top scope, так и в теле функции scvmax является глобальной, т. е. доступна для операций в обоих областях видимости.

Не описанные инструкцией global переменные Mx, I и x не доступны в области видимости функции scvmax, а переменная i является в ней локальной. Также видно, что фактический параметр - массив x также не доступен в области видимости этой функции т. к. в octave принято, что при запуске функции в этой области создается локальная копия фактических параметров (массив v).

Переменные, используемые в теле функции, являются локальными для функции. Переменные, определенные в элементах описания функции arg-list и ret-var также являются локальными для функции.

Итак, операции над глобальными переменными доступны из всех областей видимости, а над локальными — только внутри тех областей, где эти переменные определены. Если в некоторой области видимости нужен

доступ к глобальной переменной, то она должна быть описана в этой области с помощью инструкции `global`.

Примеры инструкции `global`.

```
global a
global a b
global c = 2
global d = 3 e f = 5
```

Отметим, что ее действие распространяется только до следующего индикатора конца инструкции — символов запятая (‘,’), точка с запятой (‘;’) или новая строка (‘\n"). Например, следующий код определяет одну глобальную переменную a и одну локальную переменную b, которым присваивается значение 1

```
global a, b = 1
```

Присвоить значение глобальной переменной с помощью инструкции `global` можно только один раз. Наприме, после выполнения следующего кода

```
global gvar = 1
global gvar = 2
```

значение глобальной переменной `gvar` останется равным 1.

Подчеркнем, что передача глобальной переменной как фактического параметра в списке параметров функции создаст ее локальную копию и не изменит ее глобальное значение.

Функция `isglobal (name)` выдаст значение `true`, если `name` является именем глобальной переменной.

Постоянные (`persistent`) переменные

Переменная, которая была описана внутри функции инструкцией persistant, сохранит свое значение в памяти между последовательными вызовами этой функции. Рассмотрим пример функции, выводящей количестов ее вызовов.

```
ybgv@ybgv-home:~> octave -q -p ~/MyOct
octave:1>
octave:1> unix ("cd MyOct;less count_calls.m")
function count_calls ()
    persistent n_calls = 0;
    printf ("'count_calls' вызвана %d раз\n",
    ++n_calls);
    whos
endfunction
ans = 0
octave:2> whos
Variables visible from the current scope:

variables in scope: top scope

  Attr   Name        Size                     Bytes  Class
  ====   ====        ====                     =====  =====
         ans         1x1                          8  double

Total is 1 element using 8 bytes

octave:3> for i = 1:3, count_calls, endfor
'count_calls' вызвана 1 раз
Variables visible from the current scope:

variables in scope: count_calls: /home/ybgv/MyOct/count_calls.m
```

```
  Attr   Name         Size                     Bytes  Class
  ====   ====         ====                     =====  =====
      p  n_calls      1x1                          8  double

Total is 1 element using 8 bytes

'count_calls' вызвана 2 раз
Variables visible from the current scope:

variables in scope: count_calls: /home/ybgv/MyOct/count_calls.m

  Attr   Name         Size                     Bytes  Class
  ====   ====         ====                     =====  =====
      p  n_calls      1x1                          8  double

Total is 1 element using 8 bytes

'count_calls' вызвана 3 раз
Variables visible from the current scope:

variables in scope: count_calls: /home/ybgv/MyOct/count_calls.m

  Attr   Name         Size                     Bytes  Class
  ====   ====         ====                     =====  =====
      p  n_calls      1x1                          8  double

Total is 1 element using 8 bytes

octave:4> whos
Variables visible from the current scope:
```

```
variables in scope: top scope

  Attr   Name        Size                     Bytes  Class
  ====   ====        ====                     =====  =====
         ans         1x1                          8  double
         i           1x1                          8  double

Total is 2 elements using 16 bytes

octave:5>
```

Инструкция `persistent` может задаваться в формах:

```
persistent a
persistent a b
persistent c = 2
persistent d = 3 e f = 5
```

Подчеркнем, что ни входные, ни выходные переметры функции нельзя определть как постоянные. Как и глобальные переменные, постоянная переменная может быть инициализирована только один раз. Например, после выполнения следующего кода

```
persistent pvar = 1
persistent pvar = 2
```

значение постоянной переменной `pvar` остаентся равным 1.

Если постоянная переменная не инициализирована определенным значением, она будет содержать пустую матрицу.

Значение постоянной переменной сохраняется в памяти до тех пор, пока оно не будет явно очищено инструкцией `clear`. Отметим, что постоянная

переменная удаляется из памяти только тогда, когда будет удалена функция, в тексте которой она определена.

Фиксация функций в памяти

Приоритет функций