Объекты класса Char. Функции и свойства

Объекты типа Char являются потомком ARRAY, поэтому для них справедливы все принципы, которые поддерживаются для массивов.

Объекты типа Char – строки; строки состоят из цифр, букв и символов таблицы ASCII, каждый элемент строки занимает два байта, это является нетипичным для других языков программирования, в которых одному элементу отводится один байт памяти, но MatLab ориентирован на матричные вычисления, в основе которых лежит комплексная арифметика (переход от операций комплексной арифметики к вещественной достигается автоматически с нулевой мнимой частью, обратный переход автоматически невозможен), это и обусловливает резервирование еще одного байта на комплексную часть.

Справка о создании, контроле типа и редактировании строковых переменных

```
% Задание строки:
% один или серия символов заключается в апостроф
A='ученье'
% кавычка задается четырьмя апострофами
A=''''ученье'''' % выводится  “ученье”

% контроль типа:
% является ли аргумент функции ischar строкой,
ischar(A)  % если да, то результат - логическая единица
length(A)% количество элементов строки равно 10

% Конкатенация строк (по правилом матричной алгебры):
% строки записываются матрицей размера 2×3
['tip'; 'top']
% аналогично функцией strvcat (важно: согласование размерностей по столбцам не требуется!)

% две строки последовательно записываются строкой %из шести букв,аналогично функцией strcat
['tip', 'top']
```

```
% Поиск букв или лексем:
S= 'sin(x)*cos(y)', s='x'
%  ищем в большей строке меньшую
k = findstr(s, S)
%  ищем в строке первого аргумента строку второго
r = strfind(S, s)
isempty(k), isempty(r) %  проверка хотя бы
% одного совпадения лексем (успех - логическая единица)

%% Сравнение строковых переменных v1 и v2
% v1 сравнилось с v2, если все буквы совпали с учетом
% регистра
strcmp(v1,v2)
% v1 сравнилось с v2, если все буквы совпали без учета
% регистра
strcmpi(v1,v2)
% сравнилось n букв, с учетом регистра
strncmp(v1,v2,n)
% сравниваются  n  букв без учета регистра
strncmpi(v1,v2,n)
```

Пример 1. Конкатенация (соединение) строк

```
% Конкатенация строк с пробелами и без в конце,
% первый способ:
strcat('С ', 'Новым ', 'Годом!')
% пробел – элемент строки!
strcat('С', 'Новым', 'Годом!')
% второй способ:
['С ', 'Новым ', 'Годом!']
```

Пример 2. Сравнение строк

```
% с учетом регистра и без:
s1='ABCDEFGH'
s2='ABCDefgh'
s3='ABCabc'
strcmp(s1,s2)
strcmpi(s1,s2)
strncmp(s1,s2,3)
```

Пример 3. Сравнение матриц, элементы которых - строки

```
% с учетом регистра и без:
sm1=['1234567'; 'ABCDEFG'], sm2=['1234567'; 'ABCDefg']
strcmp(sm1, sm2)
strncmp(sm1, sm2, 4)
strcmpi(sm1, sm2)
sm3=['1234567'; 'ABCDEFG']
sm4=['1234567'; 'ABCdefg']
strcmpi(sm3, sm4)
```

Пример 4. Преобразование регистра строки

```
% строчные буквы:
lower('Happy Birthday - С Днем Рождения!')
% прописные буквы:
upper('Happy New Year - С Новым Годом!')
```

Пример 5. Выделение лексем

```
% выделение слов, составляющих выражение,
% разделенных пробелами
s='С Новым Годом!'
[t1, r1] = strtok(s)
[t2, r2] = strtok(r1)
[t3, r3] = strtok(r2)
% выбраны три лексемы в переменных: t1, t2, t3

% выделение слов, составляющих выражение, разделенных
%нестандартными разделителями:
s='a+b*c'
[t1, r1] = strtok(s, '+*')
[t2, r2] = strtok(s, '+*')
[t3, r3] = strtok(s, '+*')
% второй аргумент –строка, должна содержать весь набор
%разделителей
```

Пример 6. Поиск элементов строки (подстрок в строке)

```
% результат – номер элемента в строке поиска, с которо-
го, начинаются совпадения
s1='00', s2='2003'; s3='100002'
```

```
findstr(s1,s2)
findstr(s2,s1)
strfind(s1,s2)
strfind(s2,s1)
findstr(s3,s1)
```

Пример 7. Поиск элементов (подстрок) в многомерных строках

```
sm=strvcat('com', 'compare', 'computer')
strmatch('com', sm)
strmatch('com', sm, 'exact')
s='123com'
strmatch('com',s)
sc={'com';'compare';'computer'}
strmatch('com', sc)
```

Пример 8. Поиск и замена элементов строки

```
s='12341234'
s1=strrep(s, '123', 'ABCD')
s2=strrep(s, '124', 'ABCD')
s3=strrep(s, '123', '')
```

Пример 9. Вывод элементов таблицы ASCII

```
% вся таблица
char(1:255)
% xyz
char(120:122)
```

Пример 10. Заполнение многомерной строки элементами

```
% второй и третий аргументы – размерности %массива
repmat('=', 1,4)
repmat('*-', 3,4)
```

Пример 11. Выравнивание строки

```
s=' 123456 '
sl=strjust(s, 'left')
sc=strjust(s, 'center')
sr=strjust(s, 'right')
```

Пример 12. Выполнение фрагментов строки

```
s='С Новым Годом!';
s(1:2)
s(3:8)
s(9:end)
```

Пример 13. Выявление позиций элемента в строке

```
s='С Новым Годом!';
r=eq(s, 'о')
% или равносильно:
q=r=='о';
if any(r), disp('есть совпадение'),end %
sum(r) % количество совпадений
index=find(r) % порядковые номера позиций совпадений
```

Объекты класса Cell. Функции и свойства

Cell – конструктор класса, массива разнородных объектов – ячеек. Однако с его помощью только задается размер массива. Фигурные скобки используются для перечисления его элементов, а также для указания индексов при оперировании его объектами.

Способы создания: делятся на декларативные, описательные, конвертационные и создаваемые системой. К числу создаваемых системой относятся массивы ячеек, получаемые при формировании выходных параметров переменной длины, массивы ячеек, которые используются системой и пользователем при обработке событий пользовательского интерфейса и т.д.

В отличие от ранее рассмотренных объектов, содержание массива ячеек даже при отсутствии подавления вывода точкой с запятой (;) будет невидимым.

Функция `celldisp(c)` – визуализации элементов, решит эту проблему, также как и команда c{:}

Пример 1. Создание массивов ячеек

```
A=ones(6)
% резервирование
C=cell(size(A))
b = {'sin(x.^2)/(3 * pi* x.^2) ',[1:2:pi],rand(5)}
celldisp(b)
% каждый элемент полученной матрицы –
% ячейка, состоит из одного элемента, обращение к (i,j)
% элементу %g{i,j};
g=num2cell(randn(3))
% r –массив ячеек, состоящий из одного элемента, и
% этот  элемент есть матрица класса double 4-го порядка
% – и  обращение к (i,j) элементу r{1}(i,j)
r=mat2cell(rand(4))
%понять адресацию к элементам d
d = {[1] [2 3 4]; [5; 9] [6 7 8; 10 11 12]}
iscell(d) % контроль типов
```

Пример 2. Поиск совпадающих лексем с использованием массивов ячеек

```
sc1=[{'1234'}; 'ABCDEFGH']
sc2=[{'1235'}; 'ABCDefgh']
strcmp(sc1, sc2) % поиск совпадений  без учета регистра
strcmpi(sc1, sc2)
strncmp(sc1, sc2, 3) % поиск первого совпадения трех
% подряд элементов строки
```

Пример 3. Эффективного построения блочно-диагональной матрицы - blkdiag

```
% матрицы для блоков – массив % ячеек
Blocks={rand(3);randn(5);ones(4)}
% B -  блочно-диагональная матрица
B=blkdiag(Blocks{:})
```

Пример 4. Конвертирования в char

```
str = { 'Goodbye', 'cruel', 'world' }
char(str{:})
```

Пример 5. Конкатенации

```
c = { [3 4], [5 6] };
cat(1, [1 2], c{:} ) % добавление строк
cat(2, [1 2], c{:} ) % добавление столбцов
e = {}; cat(2, [1 2], e{:} )
```

Пример 6. Создания массива ячеек

```
T = cell(1,9); % резервирование
T(1:2) = { [1], [1 0] };
for n=2:8, T{n+1}=[2*T{n} 0] - [0 0 T{n-1}]; end
T{4}
```

Создание функций в Matlab

В ML для эффективного программирования используются процедуры и процедуры-функции. Каждая процедура записывается в отдельном файле с расширением *.m и имя процедуры должно совпадать с именем этого файла.

Функции и процедуры

Для создания процедур и процедур-функций используется одинаковый заголовок, но в процедуре может быть один или несколько выходных параметров

```
function [out1,out2] = myproc(in1,in2,in3)
```

а в функции только один, который вычисляется в последнем исполняемом операторе процедуры.

```
function resfunc = myfun(in1,in2,in3)
. . . . . .
resfunc=sin(in1)*in2^in3 % некоторое выражение
```

Пример 1. Процедуры

```
function [x1,x2] = quadform(a,b,c)
d = sqrt(b^2 - 4*a*c);
x1 = (-b + d) / (2*a);
x2 = (-b - d) / (2*a);
```

Обратиться к процедуре можно `[r1,r2]=quadform(1,1,1)`, используя конкретные значения входных параметров.

В MATLAB имеются встроенные функции, которые могут иметь меняющееся число входных аргументов и меняющееся число выходных параметров. Например, функция S=svd(A) вычисления сингулярных чисел матрицы A. Она может применяться в виде [U,S,V]=svd(A), когда требуется большее чис-

ло выходных параметров. Другим примером такой функции может служить функция cat(A,B) горизонтального объединения массивов A и B . Она может иметь произвольное число входных массивов, cat(A1,A2,A3,A4).

При написании собственных функций в ML существует возможность указывать переменное количество входных и выходных аргументов. Для этого предназначен массив ячеек переменной длины varargin для входных параметров и varargout для выходных. В этом случае заголовок процедуры будет иметь вид:

```
function [out1,out2,varargout] = myproc(in1,in2,in3,
varargin)
```

Такие ситуации обусловлены тем, что пользователь сам решает в каждом конкретном случае, что ему нужно на выходе, например, кроме постоянного выходного параметра вектора-решения, точность и или номер итерации.

При обращении к такой процедуре будут заданы конкретные параметры `varin1, varin2,…` и идентификаторы `varout1, varout2,…`

Неопределенность длин этих массивов ячеек накладывает дополнительную ответственность на программиста при программировании процедур. Так в момент обращения все переменные аргументы помещаются системой в `varargin`, их следует оттуда извлечь и присвоить соответствующим сущностям-переменным.

Длину массива `varargin` определяем по формуле: количество всех входных переменных (определяет функция `nargin`) минус количество постоянных входных аргументов, так же как и длину `varargout`; количество всех выходных переменных определяет функция `nargout`.

Пример 2. Тип файла – функция. Имя файла – varlist.m

```
function varlist(varargin)
   fprintf('Number of arguments: %d\n',nargin);
% nargin – количество входных аргументов в функции
   celldisp(varargin)
```

Вызов функции:

```
varlist(ones(2),'some text',pi)
```

Результат:

```
Number of arguments: 3
varargin{1} =
     1     1
     1     1
varargin{2} =  some text
varargin{3} = 3.1416
```

Пример 3. Тип файла – функция. Имя файла – sizeout.m

```
function [s,varargout] = sizeout(x)
nout = max(nargout,1) - 1;
% nargout – количество выходных аргументов функции
s = size(x);
for k=1:nout
   varargout{k} = s(k);
end
```

Вызов функции:

```
[s,rows,cols] = sizeout(rand(4,5,2))
```

Результат:

```
s =      4     5     2
rows =     4
cols =       5
```

Пример 4. Количество входных параметров.
Тип файла – функция. Имя файла – testarg1.m

```
function c = testarg1(a,b)
if (nargin == 1)
    c = a.^2;
elseif (nargin == 2)
    c = a + b;
end
```

Вызовы функции:

```
estarg1([1 2])
testarg1([1 2],[3 4])
```

Результат выполнения:

```
ans =     1     4
ans =     4     6
```

Пример 5. Суммирование объектов double в массиве ячеек `varargin`

```
function s = add(s,varargin)
for n = 1:nargin-1
s = s + varargin{n};
end
```

Пример 6.
О массиве ячеек `varargin` входных параметров переменной длины

```
function b = blue(varargin)
if nargin < 1
varargin = {'rgb'};
end
switch(varargin{1})
case 'rgb'
b = [0 0 1];
case 'hsv'
b = [2/3 1 1];
otherwise
error('Цветовая модель не определена')
end
```

Аноним и функция-строка

Помимо описанных конструкций в ML используются анонимы. Это не-поименованные процедуры-функции одного или нескольких аргументов. Синтаксис анонимов сводится к выражению, левая часть которого является

именем процедуры, правая состоит из определяющего символа @, после которого в круглых скобках перечисляются один или несколько аргументов функции, а затем приводится её аналитическое представление, зависящее от этих аргументов, например,

```
sincos = @(x) sin(x) + cos(x);
w = @(x,t,c) cos(x-c*t);
```

Заметим, что анонимы могут быть аргументами функций, например, fzero

```
fzero( @(x) sin(x)+cos(x), 0 ).
```

Анонимную функцию можно определять прямо в командной строке ML или в пределах функции или скрипта. То есть, можно создать простые функции без необходимости создания файла специально для них.

Конструкция `inline` также обеспечивает быстрое создание функции одной или нескольких переменных в соответствие с предлагаемым синтаксисом:
`Namefunction=inline(expression_string)`

> Пример 5. Процедура `inline`
>
> ```
> g=inline('2*cos(x)-sin(y)')
> g(pi/8,pi/12)
> symvar(g) % массив ячеек, содержит аргументы функции
> g{1},g{2} % аргументы
> ```

Подпроцедуры

Помимо функций и процедур иногда целесообразно определить функцию, которая нужна только для выполнения конкретной процедуры, тогда она должна быть записана в том же файле, что и головная процедура, и является подпроцедурой (подфункцией). Подпроцедура «невидима» для остальных программ или процедур.

> Пример 6. Процедуры и подфункции
>
> ```
> function [x1,x2] = quadform(a,b,c)
> ```

```
d = discrim(a,b,c);
x1 = (-b + d) / (2*a);
x2 = (-b - d) / (2*a);
end % quadform()

function D = discrim(A,B,C)
D = sqrt(B^2 - 4*A*C);
end % discrim()
```

Практика 4

1. Дано слово s1. Получить слово s2, образованное нечетными буквами слова s1.

2. Дано слово s. Получить слово t, получаемое путем прочтения слова s начиная с его конца.

3. Дано предложение. Определить, сколько в нем гласных букв.

4. Даны два слова. Определить, сколько начальных букв первого слова совпадает с начальными буквами второго слова. Рассмотреть случай, что слова разные.

5. В предложении найти самое длинное слово. Вывести его.

6. Текст набран полностью прописными русскими буквами. Заменить все прописные буквы, кроме букв, стоящих после точки, строчными.