MATLAB Branching statements

This is Chapter 3 of the textbook

Disclaimer: *These abbreviated notes DO NOT substitute the textbook for this class. They should be used IN CONJUNCTION with the textbook and the material presented in class. If there is a discrepancy between these notes and the textbook, ALWAYS consider the textbook to be correct. Report such a discrepancy to the instructor so that he resolves it. These notes are only distributed to the students taking this class with A. Gerbessiotis in Fall 2013 ; distribution outside this group of students is NOT allowed.*

6.1.1. Logical operators and functions. We have introduced and discussed in Subject 3, MATLAB's logical operators including &, |, `xor`, $\sim$. In addition to these operators for LOGICAL-AND, LOGICAL-OR, LOGICAL-XOR, LOGICAL-NOT, MATLAB makes available a variety of logical functions. These include

```
ischar(b)     : Is b a character? If yes returns true or false otherwise.
isempty(b)    : If array b is empty return true, else return false.
isinf(a)      : If value a is infinity return true, else return false.
isnan(a)      : If value a is Not-a-Number return true, else return false.
isnumeric(b)  : If value b is numeric array,return true, else return false.
logical       : convert numerical non-zero values to true, zero value to
                false.
```

If it is not clear why one would want to use `logical` the following example might resolve such questions. In the first instance `x` is stored as a `double`, i.e. in 8 bytes, whereas in the second case `x` is stored in a single byte. Thus the use of `logical` can slash memory usage by a factor of 8.

```
>>  x=1;
>>  whos x;
>>  x=logical(1);
>>  whos x;
```

6.1.2 Reminder for $==$ and $\sim=$. MATLAB and computers in general are not as accurate as humans are. Because numbers are represented with a finite number of bits (that for most purposes this number is no more than 64 bits), several real numbers are not representable in computer memory and thus approximated by some other (i.e. different) real numbers. Thus for a human $\sin(\pi)$ is equal to 0; but for MATLAB and a computer $\sin(\pi)$ might be a very small number THAT IS NOT ZERO.

(c) Copyright A. Gerbessiotis. All rights reserved.

6.2.1 The `if-elseif-else` statement. Its syntax is highlighted below. In an `if` statement, the logical expression `control_expr_1` is evaluated first. If it evaluates to a value equal to `true`, then the sequence of `Statement 1, Statement 2` are executed and then control is transferred to the statement after the `end`. Otherwise, if `control_expr_1` is evaluated to a value equal to `false`, then similarly `control_expr_2` is evaluated. If it is true, `Statement 3, Statement 4` are executed and then control passes to the statement after the `end`. If every `control_expr` is `false` then the `else` statements `Statement 5, Statement 6` are executed.

6.2.2 Notes on `if-elseif-else`. There can be **0 or more** `elseif` statements, but there can be only at **most one** `else` statement.

```
if control_expr_1
   Statement 1
   Statement 2
elseif control_expr_2
       Statement 3
       Statement 4
else
       Statement 5
       Statement 6
end
```

6.2.3 Example. Say we want to compute y so that `y=abs(x)` . One way to do it is by

```
1 if (x>=0)    % you can  have a semicolon (;) after the condition or not
2    y=x;      %<--- This is a comment
3 else         % File sub6p3bif.m contains this example
4    y=-x;
5 end
```

One can also write the whole statement in a single line as in,

```
if (x>=0); y=x; else y=-x; end
```

(c) Copyright A. Gerbessiotis. All rights reserved.

6.2.4 Example. In this example we are given a quadratic equation and we want to figure out if it has a double root, two real (distinct) roots, or no real roots (i.e they are complex roots). For this we need to compute its discriminant and figure out whether it is zero, positive, or negative respectively. This is `sub6p4ifex.m` file. The next two files show how one can find the absolute value of x.

```
% sub6p4ifex.m
a=input('a of ax**2 +bx+c ');
b=input('b of ax**2 +bx+c ');
c=input('c of ax**2 +bx+c ');
d=b*b-4*a*c ;
if (d==0) ;
   fprintf('%f x**2 + %f x + %f  has one double real root decause Delta=%f\n',a,b,c,d) ;
elseif (d<0) ;
   fprintf('%f x**2 + %f x + %f  has no real root (i.e. it has complex roots) because Delta=%f\n',a,b,c,d) ;
else
   fprintf('%f x**2 + %f x + %f  has two distinct real roots because Delta=%f\n',a,b,c,d) ;
end
```

```
x=input('Give scalar value for x ');    % sub6p4a.m , y=absolute_value(x)
y=x;
if (x < 0)
   y=-x;
end
disp(['Absolute value of x is ' , num2str(y)]);
```

```
x=input('Give scalar value for x ');    %sub6p4b.m
if (x < 0)
   y=-x;
else
   y=x;
end
disp(['Absolute value of x is ' , num2str(y)]);
```

(c) Copyright A. Gerbessiotis. All rights reserved.

```
x=input('Give scalar value for x ');   %sub6p4c.m
if (x < 0); y=-x; else y=x; end
disp(['Absolute value of x is ' , num2str(y)]);
```

```
x=input('Give scalar value for x ');   %sub6p4d.m
y=abs(x);
disp(['Absolute value of x is ' , num2str(y)]);
```

(c) Copyright A. Gerbessiotis. All rights reserved.

More examples with nested if statements

The first segment print Head or Tails depending on whether a value is 0 or 1. A standard `if elseif else` is used in it. In the alternate implementation a `nested if` exists within the `else` statement of the outer `if`.

```
x=input('Enter value ');  % Standard if elseif else
if (x==0)
 disp('Heads');
elseif (x==1)
 disp('Tails');
else
 disp('error');
end
```

```
x=input('Enter value ');  % Nested if
if (x==0)
 disp('Heads');
else
 if (x==1)
   disp('Tails');
 else
   disp('error');
 end
end
```

(c) Copyright A. Gerbessiotis. All rights reserved.

The `switch` statement.

The switch statement is another form of branching that allows a programmer to select a specific action depending on the value of an integer variable, a character or logical expression. In such cases, when one has to explore several possibilities, one wants to avoid the use of multiple (and sometimes confusing) `if-elseif-end` statements.

For example say you want to print in english a month that is represented by an integer in the range 1-12. If `month == 1` then you want `January` to be printed, and so on. A compact way to achieve this is by using a switch statement.

```
1  month = input('input a number 1-12 for a month ');        % File : sub6p5switchex.m
2  switch (month)
3  case 1
4    fprintf(' January \n');
5  case 2
6    fprintf(' February\n');
7  case 3
8    fprintf(' March   \n');
9  case 4
10   fprintf(' April   \n');
11 case 5
12   fprintf(' May     \n');
13 case 6
14   fprintf(' June    \n');
15 case 7
16   fprintf(' July    \n');
17 case 8
18   fprintf(' August  \n');
19 case 9
20   fprintf(' September\n');
21 case 10
22   fprintf(' October \n');
23 case 11
24   fprintf(' November\n');
25 case 12
26   fprintf(' December\n');
27 end
```

(c) Copyright A. Gerbessiotis. All rights reserved.

```
1 % 1. General syntax of case statement     File: sub6p6switch.m
2 switch (switch_expression)
3  case case_expr_1
4    Statement 1;
5    Statement 2;
6  case case_expr_2
7    Statement 3;
8    Statement 4;
9  otherwise
10   Statement 5;
11   Statement 6;
12 end
13
14 % 2. Several case statements can be combined together; if they execute the same
15 %    Statement 1, Statement 2
16
17 switch (switch_expression)
18  case { case_expr_1 ,  case case_expr_2 }  % These are braces! not parentheses in the case
19   Statement 1;
20   Statement 2;
21  otherwise
22   Statement 5;
23   Statement 6;
24 end
```

6.3.1 Syntax. If the value of `switch_expression` is equal to `case_expr_1`, then `Statement 1, Statement 2` are executed. If the value of `switch_expression` is equal to `case_expr_2`, then `Statement 3, Statement 4` are executed. For any other value not equal to the previous ones considered an optional `otherwise` statement might become available that catches all other values; in such a case `Statement 5, Statement 6` are executed. If there was no `otherwise` statement, other values would have been ignored and control would have passed to the statement immediately following the `end`. This is shown with the standard syntax of `switch` in lines 2-12. If several case statements seem to execute identical code, then they can be combined together in the same case statement, separated by commas as shown in lines 17-24 above.

(c) Copyright A. Gerbessiotis. All rights reserved.

Two more examples with the `switch` statement.

```
digit = input('decimal digit ');  % sub6p7aswitchex.m
switch (digit)
case { 1 , 3 , 5 , 7 , 9 }
   fprintf(' Odd  digit %d \n',digit);
case { 0 , 2 , 4 , 6 , 8 }
   fprintf(' Even digit %d \n',digit);
end
```

```
digit = input('decimal digit ');  % sub6p7bswitchex.m
switch (digit)
case { 1 , 3 , 5 , 7 , 9 }
   fprintf(' Odd  digit %d \n',digit);
otherwise
   fprintf(' Even digit %d \n',digit);
end
```

```
digit = input('decimal digit ');  % sub6p7cswitchex.m
switch (digit)
case { 1 , 3 , 5 , 7 , 9 }
   fprintf(' Odd  digit %d \n',digit);
case { 0 , 2 , 4 , 6 , 8 }
   fprintf(' Even digit %d \n',digit);
otherwise
   fprintf(' Unrecognized digit %s\n',digit);
end
```

(c) Copyright A. Gerbessiotis. All rights reserved.

The syntax of the `try/catch` statement.

```
try                                          % sub6p8trycatch.m
  Statement 1;
  Statement 2;
catch
  Statement 3;
  Statement 4;
end
```

6.4.1 `try/catch` syntax. Whenever a `try` keyword is reached, the statements `Statement 1, Statement 2` within it are executed in the order given. If they generate NO errors, the catch statements are ignored and control is passed to the statement following the `end` of `try/catch`. Otherwise, if `Statement 1, Statement 2` generate an error (not a warning!), the `catch` statements are executed immediately following the Statement that generated the error. Thus `Statement 3, Statement 4` can be used to recover from the error or allow for a graceful exit from that encountered error.

(c) Copyright A. Gerbessiotis. All rights reserved.

An example with the syntax of the `try/catch` statement.

```
1 % naive way to do things : sub6p9atrycatchexno.m
2 %
3   monthlist= ['January  '; 'February '; 'March    '; 'April    '; 'May      ';
4               'June     '; 'July     '; 'August   '; 'September'; 'October  ';
5               'November '; 'December ' ];
6   month = input('input a number 1-12 for a month ');
7   month_name =monthlist(month,1:end);
8   fprintf(' Month is %s \n',month_name);

1 % robust way to do things   sub6p9btrycatchex.m
2 try
3   monthlist= ['January  '; 'February '; 'March    '; 'April    '; 'May      ';
4               'June     '; 'July     '; 'August   '; 'September'; 'October  ';
5               'November '; 'December ' ];
6   month = input('input a number 1-12 for a month ');
7   month_name =monthlist(month,1:end);
8   fprintf(' Month is %s '); disp(month_name); fprintf(' \n');
9 catch
10  fprintf('\n\nException! Month value   %d given as input  is not in the range 1-12\n',month);
11 end
```

6.4.2 Example. In this example we try to do the same thing we did with the `switch` statement of section 6.3.1, except that we use a column vector of strings to store the 12 months. In addition we do some kind of rudimentary input checking. The only array indices that are valid are 1-12; however a careless or malicious user might input an out-of-range value through line 6 that will cross the array bounds when it accesses `monthlist` in line 7. Thus an error might get generated in line 7; if not, line 8 will print the month with the numeric value provided in line 6. If an exception occurs, line 10 of the `try-catch` version is executed to tell us that something went wrong.

As a side-note, you might observe that lines 8 of both programs are not identical; one uses the `disp` function, the other does not. This is to show multiple ways of doing the same thing.

(c) Copyright A. Gerbessiotis. All rights reserved.

More if-then-else examples

```
x = input('␣Enter␣Number␣');                    % Absolute value example
if (x >=0) ; y=x; else y=-x; end                % Using an if-then-else statement
fprintf('Absolute␣value␣of␣%f␣is␣%f␣\n',x,y);
```

```
x=input('digit␣');                              % sub6p10odd1.m
if (x == 1 || x== 3 || x==5 || x== 7 || x== 9)  % Simple if-then-else
  fprintf('␣odd␣␣%d\n',x);
else
  fprintf('␣even␣%d\n',x);
end
```

```
x=input('digit␣');                              % sub6p10odd2.m
if (x<0 || x > 9)                               % Rudimentary input checking
  fprintf('␣%d␣is␣not␣a␣single-digit␣number␣between␣0␣and␣9␣(inclusive)␣\n',x);
elseif (x == 1 || x== 3 || x==5 || x== 7 || x== 9)
  fprintf('␣odd␣␣%d\n',x);
else
  fprintf('␣even␣%d\n',x);
end
```

```
x=input('digit␣');                              %sub6p10odd3.m
if (x<0 || x > 9)                               %Nested if statement
  fprintf('%d␣is␣not␣a␣single-digit␣number␣between␣0␣and␣9␣(inclusive)\n',x);
else
if (x == 1 || x== 3 || x==5 || x== 7 || x== 9)  % This if-statement
  fprintf('␣odd␣␣%d\n',x);                      % is inside the else statement of the
else                                            % outside if-statement
  fprintf('␣even␣%d\n',x);
end
end
```

(c) Copyright A. Gerbessiotis. All rights reserved.

More switch-case-otherwise examples

```
x=input('digit ');          %sub6p11oddswitch1.m
switch (x)                  %Simplistic use of switch-case
case 0
  fprintf('even \n');
case 2
  fprintf('even \n');
case 4
  fprintf('even \n');
case 6
  fprintf('even \n');
case 8
  fprintf('even \n');
case 1
  fprintf(' odd \n');
case 3
  fprintf(' odd \n');
case 5
  fprintf(' odd \n');
case 7
  fprintf(' odd \n');
case 9
  fprintf(' odd \n');
otherwise
  fprintf(' invalid \n');
end
```

```
x=input('digit ');          %sub6p11oddswitch2.m
switch (x)                  %Compact use
case { 0,2,4,6,8}
  fprintf('even \n');
case {1,3,5,7,9}
  fprintf(' odd \n');
otherwise
  fprintf(' invalid \n');
end
```

(c) Copyright A. Gerbessiotis.

More switch-case-otherwise examples

```
x=input('digit ');                % sub6p11oddswitch3.m
y=-1;                             % A complex use of switch
switch (x)                        % y can have one of three values -1,0,1
case { 0,2,4,6,8}                 % when printed through the last line.
  y=0;
  fprintf('even \n');
case {1,3,5,7,9}
  y=1;
  fprintf(' odd \n');
end
  fprintf(' y= %d\n',y);
```

More examples are available in the protected area of the course web-page through link C3.ii of Section C3 in subdirectory Sub6.

(c) Copyright A. Gerbessiotis. All rights reserved.