'while LOOPS'

% When the programmer needs finer control on the looping process the MATLAB 'while loop' is used which starts the loop with the key word '**while**' similar to C++ and ends the loop with an '**end**' statement. The while loop is a conditional loop in that it depends upon a logical condition staying true for repetition of the body of the loop to continue as in C++. Logical conditions are statements that embody the use of logical operators we encounter in mathematics. Consider the following interactive example which contains the logical condition '**n<3**' (Is the value of n less than 3? Answer is either true or false depending on n, but n will change in the loop which will stay active till the condition is false, more details below)

```
» n=0;
» while n<3
  disp('My friends')
  n=n+1;    % loop variable  "n"  must change to control the loop!
 end;
```

```
My friends
My friends
My friends
```

%**what is the value of n when it ends?**

% In this last example the variable '**n**' controls the number of times the body of the loop will be repeated. Each time the computer passes through the loop the value of n increases by 1 until it equals 3 at which time the while loop will terminate because the logical statement n<3 is no longer true. The computer will then seek out any statements beyond the 'end' line (none in this example).

Playing Computer

% It is helpful when first learning programming to follow the program variables, especially the one that is controlling the flow of the program like "n" in this while statement. One way is to set up a table of its values, outputs and logical conditions and note the changes line by line as the computer is executing the statements. In the example above, we might construct the following table that follows the action. Examine the program line by line above and 'play computer' (You figure out what the computer will do) as you study this table.

n	n<3	outputs
0	true	My friends
1	true	My friends
2	true	My friends
3	false	% loop terminates

% In fact, in any complex situation this process of following the variables (All of them if needed!), as well as, any outputs by the program can help debug problems, in addition, to understanding the overall logic followed by the computer system.
% In general, when you study an example or problem below and need to understand it better, then set up similar tables to help your comprehension of what is taking place as the program executes..
% NOTES: In this last example the fact that the control variable 'n' changed in the body of the loop was due to the statement **n = n+1, a simple counter,** eventually terminated the loop. It is very important to realize that something must change the logical condition in the body of the while loop else the condition will always stay true (or false in other situations). If the while condition stays true always then the computer will not exit loop. We say under these circumstances that we have an **infinite** loop. In the above program if we left out the n=n+1 statement we would have output of "My friends", theoretically forever. However, if you find you are in an infinite loop just hit the "ctrl' and "c" keys to stop the infinite loop.
% So bear in mind that within the body of the while loop, logic dictates that we have lines that will

change the controlling logical statement(s). Usually, the statement just before the end statement, is the one that effects the loop logic.
 % Consider the next program example (script file average.m) that uses an input statement to control the number of repetitions of the loop. The logic statement used to control the while loop x ~= 0 means the value of x is not equal to 0. Here x is zero only when the user enters a zero and that terminates the loop. Ie. A special value to terminate the loop.

DEMO

```
» average.m
% This program will ask a user for values and when the user is finished
% produce the average value of all the values entered.
% The program does not know how many values are to be given, so it counts them!
% The program first gives some instructional information to a user!
disp('AVERAGE CALCULATION PROGRAM');
disp('Please enter the values you want to average when you see the prompt');
disp(' "VALUE ?". Enter a ZERO (0) when you have finished.');
count= 0;
sum = 0;
x=input('VALUE ?  ');
while x ~= 0
  sum = sum +x;
  count= count +1;
  x=input('VALUE ?  ');
end
disp ('The AVERAGE of all your VALUES is');
disp (sum./count);
```

Executing the program we get

```
» average
AVERAGE CALCULATION PROGRAM
Please enter the values you want to average when you see the prompt
 "VALUE ?". Enter a ZERO (0) when you have finished.
 VALUE ?  20.5
 VALUE ?  30.5
 VALUE ?  15.5
 VALUE ?  0
The AVERAGE of all your VALUES is
   22.1667
```

%The program assumes a person knows that they must hit the enter key after each of their values (20.5 30.5 15.5 and 0).
The counter 'num' increased by one each time a value was entered with the statement num=num+1, hence, it kept track of the number of values.
The 'sum' variable kept adding the values obtained from the input statement placed into the 'x' variable with the statement sum = sum + x within the body of the while loop.
Finally to get an average was easy since we had the sum of all values and a count of all values and by definition the average = sum/num.

We can follow the action of 'average.m' by 'playing computer' as follows

sum	num	x	output
0	0	20.5	
20.5	1	30.5	
51.0	2	15.5	
66.5	3	0	

  22.1666666666666668 if “format long” is still active in memory or
 22.1667 if “format short” is active
% The output was simply 66.5/3.

LOGICAL OPERATORS AND STATEMENTS

% In the **while** statement and the soon to be introduced **if** statement we need logical statements for control. MATLAB provides the following logical operators. There meaning is also listed. Look over this list carefully it should look familiar.

Relational Group: The operators <, <=, ==, >, >= are identical to C++ but ~= is equivalent to != in C++and the **relational operators** are similar to the inequality statements of algebra.

Logical Group: The operators are similar to C++ but are &, | and ~ . called, “and”, “or” and “not” respectively. These help us construct simple and complex logical conditions. These are called the **logical operations** in MATLAB.

OPERATOR	**MEANING**
<	**Less than**
<=	**Less than or equal**
==	**Equal (be careful not to use = in logical statements)**
>	**Greater than**
>=	**Greater than or equal**
~=	**Not equal**
&	**“and” used for element by element arrays**
&&	**“and” for scalars in conditional expressions**
|	**“or” for arrays**
||	**“or” for scalars in conditional expressions**
~	**not**

% Logical statements are evaluated until they are equal to the values of true or false. This is actually done mathematically so that the value of true is equal to 1 and false is equal to 0. We will now construct a number of logical statements to determine if they are false or true. We begin with simple statements and advance to more complex logical structures to illustrate the principle. First we set values to variables then we interactively let MATLAB evaluate the logical structure. MATLAB will put the answer in the special variable 'ans' since we are not putting the answer into a variable. If ans =1 then the statement is equivalently true. If ans =0 then it false. To save space we put MATLAB's response on the same line as the logical statement. Study each of the following.

% **Simple logical statements:**

```
» p = 5;
» q = 8;
» p < q       ans =    1
» q < p       ans =    0

» p = 5;
» q = 5;
» q < p       ans =    0
» q <= p      ans =    1
» q ~= p      ans =    0
» p == q      ans =    1
» p >= q      ans =    1
» p == 5      ans =    1
» p == 4      ans =    0
```

```
% multiple logical constructions
using the logical operators &&    ||    ~
» r = 9;
» s = 3;
» p = 5;
» q = 4;
```

% The 'and' operator leads to truth only if both simple statements are true. Any other combination is false.

```
» p < q &&  r >= s   ans =     0     %Here p<q is false hence its all false
» q < p &&  r >= s   ans =     1
```

% The 'or' operator leads to truth if any one of its simple statement is true. It leads to false only if both **operands** (statements on each side of operator) are false.

```
» q < p || r >=  s   ans =     1
» p < q || r >= s    ans =     1
» p < q ||  s >= r    ans =     0
» ~(p < q) || s >= r ans =     1
```

% Note, in the last example, the use of the 'not' operator to reverse the logic of the p<q statement, making it true and ultimately the whole ‘or’ statement since only one part has to be true to make the whole true.

» In general it is a good idea not to let your logic statements get too complicated since this makes it difficult to fix problems. It is proper use of these statements that adds intelligence to our program. Generally the problem at hand will define for us the framework under which we have to construct our program. This framework will suggest what variables are used for loop control as well as the ability to select statements from several alternate ones (The topic of the selection **“if”** command which follows).

SELECTION PROGRAMMING

% A great deal of the intelligence of a program is obtained by having commands that permit the programmer to choose statements to be executed that depend upon logical conditions. This concept is implemented in MATLAB (and most computer languages) with the **if** command and its associated **else** statement. We say that the **if** statements give us **conditional control on the flow** (the order statements are executed). The **if** command and associated statements have several variations and in combination with repetition statements bring out the true power of a computer. We will now examine these variations and combinations.

I: SINGLE TRUE LOGICAL CONDITION (simple if)

% We can set up an **if** with one logical condition to execute a single or multiple commands only if the logical condition is true.

In interactive mode or in a program we can use, after we set the value of x, the following **if** form to execute a single statement.

```
» x=3;
» if x==3, disp('x is OK'); end
x is OK
```

% MATLAB executed the disp() function because x equals 3.

% We note that to terminate an **if** statement an **end** statement is needed similar to the **for** and **while** loops. But the **if** statement is **not a loop,** it is executed **only once**. It can be placed within a loop if repetition is wanted.

```
% another example
» x=8;
» if x < 9, y=x+99; end;
» y                          % We create the value of y after the if
y =   107
```

%Each if statement permits the execution of a statement(s) in it only **if** the logical condition is true. Which in this case it is true x<9 and the statement y=x+99 is executed. If x<9 was false the y=x+99,end would not be executed and in this case if we ran the above with x=10 to make the latter false y which was not initialized would have been reported as "undefined" when we attempted. >>y
% It is better in a program to show by **indentation** the command or commands that will be executed for a true condition in the **if** statement, as in the next program.
(this indentation here and in "for' and "while" statements also helps if there are "bugs" in our program since it makes it easier to see the logic of the program.

DEMO
```
%if2.m
x=8;
» if x ~= 12
   disp('Your choice was correct');
   y =x.^2;
   disp('and here it is');
   disp(y);
 end
```

% The condition is true and MATLAB executes all statements between the if line and the end statement as;

```
Your choice was correct
and here it is
   64
```
% Only if x ~= 12 is true will the above commands execute.

% To see that the disp() statement can also not be executed in the last example suppose we set x=12 and do the following;
```
»x = 12;
» if x ~= 12
   disp('I don''t think I will see this line');
 end
```

% The display function line did not appear after execution because the condition x~=12 was false since x is equal to 12.
%NOTE: The special use of '' above to get an output of a single ', because in the disp() function the single quote, ', has a marker meaning. THIS is not a double quote but two single ones.
TRY the following to see this in action.

```
»disp('I don''t think I will see this line, WHAT"S UP!');
```
Then also with two single quotes at the end.
```
»disp('I don''t think I will see this line, WHAT''S UP!');
```

DID THIS LATTER WORK..WHY OR WHY NOT??

II: A SINGLE TRUE OR FALSE OPTION IF-ELSE

% When the **if** statement is **true** we saw that we can execute commands under its control, but, many times, we want an alternative set of commands to execute when the logical condition in the **if** statement is **false**. MATLAB as well as, many other computer languages, provides the connecting statement known as the **else** statement to achieve this last point. The **else** statement is actually part of the original **if** statement **and does not stand by itself.**
We illustrate the use of the **else** extension of the **if** statement. **Study these examples carefully.**
% Consider a value of x that is valid only in a range, as in;
% Note tan(x) function, x is in radians in Matlab

```
%else1.m
 A =input(‘Enter the  angle for the tan function in degrees?  );
   x=a*pi/180;
   if x ~=pi/2
      y= tan(x);
      disp(‘the tangent of your angle is’);
      disp(y);
  else
      disp('The angle you specified is not valid for this tangent function');
  end
```

```
»else1
Enter the angle for the tan function in degrees?   90
The angle you specified is not valid for the tangent function
```

% Here the command between the **else** and the **end** belongs to the false branch of the original **if** statement. In this last case the logical condition was false and the computer executed the disp() function. The opposite situation is illustrated with angle assigned within the program;

```
%else2.m
 x=pi/8;
 if x ~=pi/2
    y= tan(x)
  else
   disp('The angle you specified is not valid for the tangent function');
  end
```

```
»else2
y =   0.4142
```

% MATLAB just evaluated the tan(pi/8) and produced the value(notice that the line y=tan(x) did not have a ';' at its end to produce the value of y).

% More than one command can be executed in each branch of this if-else combined statement with a more complex logic decision

```
%else3.m
 x=3.4;
  if x > 4.0 & x< 10.0
     y=5.0.*x.^2
     disp ('The value calculated for y is parabolic solution”)
  else
      y= 8.0.*x -7
      disp( 'The value of y  calculated is linear')
  end
```

```
»else3
y =   20.2000
The value of y is calculated is linear
```

% Note: the logical condition stated confines x to a range from 4 to 10 for the parabolic formula branch and anything else is evaluated from the linear formula.

III: SELECTION PLUS REPETITION

```
>>type pigw     % used if it's in your work place.
%pigw.m
% This program computes the selling price of pigs.
disp('          Pig Evaluation Program');

weight = input('Enter the Pigs weight in pounds ( Use a ZERO to exit ) ');
while weight ~= 0
   if weight > 300
      disp(' The pig is ready for the Market');
      price = 0.25.*weight;
      disp(' The dollar value of the pig is');
      disp(price);
   else
      disp('Feed the pig more corn and return next month');
   end
   weight =input('Enter the Pigs weight in pounds ( Use a ZERO to exit)');
end
```

% NOTE: 1. With in a while loop is an if-else combination.
 2. Indentation for blocks of code make it clear what is under the control of the **while** loop and both the **if** and **else** statements.
 3. Note the placement of the two **end** statements. Matching positions in the program while-end and if-end.
 4. The control variable **weight** for the while loop is **initialized before the loop** and **must be changed near the end of the loop** by input so that the program will continue until the exit value "0" is entered.
 5. Note comments to the programmer and output comments to user.

% The program is now illustrated in a typical execution.

```
» pigw
          Pig Evaluation Program
 Enter the Pigs weight in pounds ( Use a ZERO to exit ) 400
  The pig is ready for the Market
  The dollar value of the pig is
   100
 Enter the Pigs weight in pounds ( Use a ZERO to exit ) 150
Feed the pig more corn and return next month
 Enter the Pigs weight in pounds ( Use a ZERO to exit ) 0
```

MULTIPLE LOGICAL OPTIONS(increasing the complexity of options)

% By using additional statements known as the **elseif** statement attached to the **if** statement we can set up a number of logical alternative paths for the computer to take. Each path is taken based upon the logical evaluation of a condition.
Consider the following program illustration of alternative paths There are five possible paths the program can take depending on two parameters, the last is to exit the while loop with both values =0.. **Study this code to see the paths..**

DEMO

Run for n=-9,p=90; n=9,p=78; n=p=40; & n=45 p=12; both =0

```
%demoelseif.m
% calculating a jet engine thrust for parameter models
  n=input('parameter one for engine thrust');
  p=input('parameter two for engine thrust');
  while(n~=0&& p~=0)
   a=98.00;
   x=.2;
   if n<0 && p>0
     x=.9/p;
     y= a*x^2;
     disp('case 1');

   elseif p>n && n>0
     x=6.*x + n*p;
     y=a/2*(x+500);
     disp ('case 2');
   elseif n == p
     x=90*x;
     y=a*p;
     disp ('case 3');
   else
     y=0;
     disp ('case 4')
     disp('bad parameters, redue your model');
  end;
  disp('for your parameters');
  disp([n p]);
  disp('maximum engine thrust is in pounds');
  disp(y)
  disp('          ');
  n=input('parameter one for engine thrust');
  p=input('parameter two for engine thrust');
 end
  disp ('that''all folks');
```

% Once we set up a pattern, as above, only one path through the if will be followed by the computer. The first path that is true will cause the execution of the statement(s) under its control. Thus, in the previous example if n<0 was not true the first elseif with p>0 & q==p would be tried next. If this first elseif was not true the next elseif would be evaluated and so forth. More elseif statements could be attached. The final statements under the control of the **optional else** statement would be executed only if all other alternative paths were false

% To illustrate an application of multiple alternatives, we will build up a simple program into more complex forms, as well as, introduce a few new MATLAB functions on the way. That is, we will start with the basic mathematical solution and embellish it with our multiple logical options to polish the programs abilities. In other words when solving real world problems we usually start out to establish the basic core solution and then once we have that we can build to expand our program and also embellish it.
% We start by considering the basic and simple script program that follows that solves for the roots of the quadratic equation which as you know has the basic mathematical form

$$x = \left(-b \pm \sqrt{b^{2-}4ac}\right)/2a$$

»type quadabc
%$x = \left(-b \pm \sqrt{b^{2-}4ac}\right)/2a$ is the standard solution of the quadratic equation. $ax^2+bx+c=0$

```
% EXAMPLE quadratic equation illustrated ax^2 + bx + c = 0
% M FILE CALLED QUADABC
disp('please enter the coefficients of the quadratic equation ');
a=input('the a= ');
b=input('the b= ');
c=input('the c= ');
% we evaluate the discriminant= b2-4ac .
clc
dis=b.^2-4.*a.*c
%possible solutions depend on the value of the discriminant
if dis>0
   disp('here are the  real root(s)');
   x1=(-b+dis.^0.5)./(2.*a)
   x2=(-b-dis.^0.5)./(2.*a)
elseif dis==0
    disp('There is only one root');
    x=(-b+dis.^0.5)./(2.*a)
elseif dis<0
   disp('The roots are imaginary and are ');
   x1=(-b+dis.^0.5)./(2.*a)
   x2=(-b-dis.^0.5)./(2.*a)
end
```

% The final root values x1 and x2 are output to the screen from the MATLAB echo of variable values x1 and x2 when we do not have semicolons at the end of the line.
% Note the use of **clc** whose meaning we get from MATLAB's **help** command.

```
» help clc
clc      clears the command window and homes the cursor..
```

% We now execute the quadratic root program

```
» quadabc
please enter the coefficients
 the a= 1
 the b= 5
 the c= 1
note here the dis =  21
here are the root(s)
x1 =   -0.2087
x2 =   -4.7913
```

% We rerun the program for the interesting case of complex roots.

```
» quadabc
please enter the coefficients
 the a= 1
 the b= 2
 the c= 3
note here the dis =    -8
here are the root(s)
x1 =  -1.0000 + 1.4142i
x2 =  -1.0000 - 1.4142i
```

% NOTE MATLAB IS CAPABLE OF WORKING IN THE COMPLEX DOMAIN. This illustrated technique of getting values by leaving out the semi-colon is the simplest and easiest technique to get out the complex values. MATLAB has lots of functions for complex numbers.

fprintf() AN OUTPUT FUNCTION

% A good way to improve the output to the screen (OR LATER A FILE) is the MATLAB function 'fprintf()'. This function has a variety of uses and we will illustrate several. A typical use of fprintf() might look like

```
»xdol = 123.3455;
»fprintf('The value of our variable is %8.2f dollars.\n',xdol);
The value of our variable is   123.35 dollars.
```

NOTE: %8.2f saves 8 spaces for a fixed number with 2 decimal places as you see above placed within the output string!

The basic function **fprintf()** can be written in general form as fprintf ('format',variable list). The format part contains text similar to the display function (between single quotes) but the value of the variable (or variables) listed after the comma is placed within the text. The value of the variable seen within the text is dictated by special codes (**called conversion specifiers**) placed at the position the programmer wants the number to appear. **Each variable must have a specifier**. The specifiers within the single quotes are applied to the variables in the order they appear after the comma(do you see the comma?).
Conversion specifiers and their meaning are;
'%e' put the number out in exponential form;
'%f' put the number out in decimal form as we did above.
'%g' use decimal form for a specified range but exponential form for very large or very small numbers.
We also specify the number of positions (called the field width) for the number, as well as, the number of decimal numbers to display. In our example above '%8.2f' means to display in a field of 8 spaces a number with 2 decimals. In the example above, the value 123.35 needs only 6 positions to be displayed, the extra two positions (8 were requested) are filled with spaces on the left of the number. We say that the number is placed to the right of the field width(also called **right justification**). With these rules the programmer has almost full control on the precise way information can be displayed**. If you do not use an appropriate field width for your value, MATLAB will provide one** (See illustrations below). Both the 'f' and 'e' specifiers work as described. The 'g' specifier as in %w.dg will attempt to give priority to the number of digitsafter the decimal point, as 'd' within a field of 'w' wide.
% The special symbol '\n' within the format will advance the output position for the next command to the beginning of the next line, just like in c++. If the symbol '\n' is not in the format then the next output starts at the position we have left off on the screen. ”\n” is like using the return button on a key pad when typing text.

% DEMO demofprintf To better understand the above discussion on fprintf() we run following commands in the interactive mode.

% Carefully observe the following interactive illustrations of the fprintf() function to learn how to use it by example.

```
» xdol = 123.3455;
» fprintf('The value of our variable is %8.1f dollars \n',xdol);
The value of our variable is    123.3 dollars

» fprintf('The value of our variable is %8.0f dollars \n',xdol);
The value of our variable is      123 dollars
```

%NOTE: if the specifier has a zero for the number of decimal digits than the number is produced as an integer!

```
» fprintf('The value of our variable is %3.0f dollars \n',xdol);
The value of our variable is 123 dollars
```

```
» fprintf('The value of our variable is %1.1f dollars \n',xdol);
The value of our variable is 123.3 dollars
```

% Here MATLAB provided the appropriate space because the programmer did not!
% Multiple variable examples, study the output carefully, consider x, y and z whose values are defined as

```
» x=12.345;
» y=99.3;
» z=333456.89
» fprintf('x, y and z have values %6.3f, %5.2f & %11.4g \n',x,y,z);
x, y and z have values 12.345, 99.30 &  3.335e+005

»fprintf('x, y and z have values %6.3f, %5.2f & %11.5g \n',x,y,z);
x, y and z have values 12.345, 99.30 &      333460

»fprintf('x, y and z have values %6.3f, %5.2f & %11.3g \n',x,y,z);
x, y and z have values 12.345, 99.30 &   3.33e+005

» fprintf('x, y and z have values %6.3f, %5.2f & %11.3e \n',x,y,z)
x, y and z have values 12.345, 99.30 &  3.335e+005
```

% We next modify the quadratic program above to improve the output of the program by using the fprintf() function and handle the three possible roots that can come about, namely, two real roots, one real root, complex roots

DEMO quadabcp

```
echo off
%output via 'fprintf' function( similar to the C language function).
%first we get the coefficients
fprintf('please enter integer coefficients for each case you wish to
test\n');
a=input('the a= ');
b=input('the b= ');
c=input('the c= ');
%we evaluate the discriminant here and have MATLAB echo the results
% and clear the screen with clc command
clc
dis=b.^2-4.*a.*c
% We compute the roots and keep them in the computer's memory
x1=(-b+dis.^0.5)./(2.*a);
x2=(-b-dis.^0.5)./(2.*a);
% We do not output roots with fprintf() for complex roots but we know
the discriminant determines the three possible outcomes.
if dis < 0
   disp ('The roots are Complex and are equal to');
   x1
   x2
elseif dis==0
   % one Real roots
   fprintf (' only one Root  are x = %5.3f  \n',x1);
else
    % only dis>0 is possible here so two real roots are produced
     fprintf (' The two real roots   are x1 = %5.3f  and x2=%5.3f
\n',x1,x2);
end;
fprintf ('Your values of a,b,c  used are %3.0f, %3.0f, %3.0f\n',a,b,c);
```

```
%The function fprintf() has a lot of uses and especially when we want to put data into a file for
later use. But we will explore that later.
>> help fprintf        for now only some relevant info only
 fprintf …
   fprintf(FORMAT, A, ...) formats data and displays the results on the screen.
  FORMAT is a string that describes the format of the output fields, and
   can include combinations of the following:
      * Conversion specifications, which include a % character, a
        conversion character (such as d, i, o, u, x, f, e, g, c, or s),
        and optional flags, width, and precision fields.  For more
        details, type "doc fprintf" at the command prompt.
      * Escape characters, including:
          \b     Backspace            ''   Single quotation mark
          \f     Form feed           %%   Percent character
          \n     New line            \\   Backslash
          \r     Carriage return      \xN  Hexadecimal number N
          \t     Horizontal tab       \N   Octal number N
%We now run the program but in all cases the “clc” statement is used to see the action and logic
% and these notes remove a lot of blank regions in the output.
% we run and use coefficients that give us complex answers based on dis<0

quadabcp
please enter integer coefficients for each case you wish to test
the a= 1
the b= 2
the c= 3
dis =    -8
The roots are Complex and are equal to
x1 =  -1.0000 + 1.4142i
x2 =  -1.0000 - 1.4142i
Your values of a,b,c  used are   1,   2,   3
% We now rerun for real roots

>> quadabcp
please enter integer coefficients for each case you wish to test
the a= 1
the b= 5
the c= 2
dis =    17
 The two real roots   are x1 = -0.438  and x2=-4.562
Your values of a,b,c  used are   1,   5,   2

% We now rerun for the single real root  case

>> quadabcp
please enter integer coefficients for each case you wish to test
the a= 1
the b= 2
the c= 1
dis =     0
 only one Root  are x = -1.000
Your values of a,b,c  used are   1,   2,   1
```

```
%A better approach would be
% run this program for all cases

DEMO quadwhile
echo off
%quadwhile
```
$\%x = \left(-b \pm \sqrt{b^{2-}4ac}\right)/2a$

```
%EXAMPLE quadratic equation illustrated ax^2 + bx + c = 0
%output via 'fprintf' function( similar to the C language function).
%first we get the coefficients
fprintf('please enter the coefficients of the quadratic equation all zero’s end the
program\n');
a=input('the a= ');
b=input('the b= ');
c=input('the c= ');
while a~=0 && b~=0 && c~=0
  %we evaluate the discriminant here and have MATLAB echo the results
  dis=b.^2-4.*a.*c
  %clear the workplace
  clc
   % We compute the roots and keep them in the computer's memory
   x1=(-b+dis.^0.5)./(2.*a);
   x2=(-b-dis.^0.5)./(2.*a);
   % We do not output roots with fprintf() for complex roots but we know the discriminant
   % determines the three possible outcomes.
   if dis < 0
      disp ('The roots are Complex and are equal to');
      x1
      x2
   elseif dis==0
      % one Real roots and illustrate output with e format with fprintf.
       fprintf (' only one Root  are x = %5.3e  \n',x1);
   else
      % only dis>0 is possible here so two real roots are produced, note 2 variable output f
format.
       fprintf (' The two real roots   are x1 = %5.3f  and x2=%5.3f \n',x1,x2);
   end;
   fprintf ('Your values of a,b,c  used are %3.0f, %3.0f, %3.0f\n',a,b,c);
   % input again for the while loop
  fprintf('enter the coefficients of the quadratic equation all zero’s end the program\n');
  a=input('the a= ');
  b=input('the b= ');
  c=input('the c= ');
end
  fprintf (‘That is all Folks!’);
```

LABORATORY TASKS

Print up the final m file and outputs you collected right after each m file. Please number which problem you are submitting. Each exercise counts as a separate lab exercise.

FOUR PROCEDURES TO FOLLOW,
(not doing so will result in the loss of points for your laboratory tasks).
1. ALL OUTPUT SHOULD NOW ONLY DONE WITH fprintf(), no more display().
2. ALL CODE SHALL BE INDENTED FOR WHILE AND FOR LOOPS AND FURTHER INDENTED FOR IF-ELSE AND IF IFELSE ELSE CODE. POINTS OFF FOR NOT FOLLOWING THIS IMPORTANT PROGRAMERS STYLE.
3. USE “type filename” to display your m files you successfully debugged and ran. Copy the listing that comes out with the type command and also the “type “command that produced the listing to show me you have done an m-file to carry out a task calling for a program.
4. COMMENTS ON THE NATURE OF THE PROGRAM IN THE BEGINNING OF EACH ARE EXPECTED FOR THE PROGRAMMER AND OUTPUT SHOULD TELL THE USER WHAT THE VALUES MEAN including units. For example: velocity = 12 ft/sec

24(3 pts). Write a program that follows the flight of a golf ball, in intervals of 0.1 seconds, after it is hit. We know that its position above the ground, y, its position from the tee, x, and its velocity in the vertical direction, v, are given by the following relations with respect to the time, t, from the start of the flight. $x= 58.50t$ ft $y=83.55t-16t^2$ ft $v=83.55-32t$ ft/sec

Use a **while loop** HENCE SCALAR CALCULATIONS OF THE FORMULAS and output the values of t, x, y and v in **labeled columns,** so long as, y remains positive and the velocity stays greater than -60 ft/sec. HINTS: You may need to initialize variables to get into the loop.

25(3 pts). Write a program (script m-file) that evaluates the following naive approach to the air resistance force on a wing model in an air tunnel. The program asks a user for a velocity and produces the value of the air resistance of the wing. The program should only terminate when the user enters any negative number for the velocity. It is known that the air resistance force is given by $-Bv$ for velocities greater than or equal to 120 mph and by $-Bv^2$ for velocities that are less . The value of the constant B for this wing model has been found to be 0.039. Make sure messages for input and output from the program help make the program meaningful to a user. Add a few comments to yourself in the program. Run the program to cover all velocity ranges in the program to be sure it is working. **USE WHILE loop and IF-ELSE structure**

26(3 pts). In cosmology the behavior of space through time is given by a function called the scale factor = r(t). The variable, t, is related to time and has the range 0 to 2 pi interval 0.01. The Freedman model predicts three possible behaviors of the scale factor depending upon estimates of the amount of matter in the universe. The amount of matter depends on a factor known as **Epsilon (E)** being 0,+1 or -1. With E =0 then r(t) =Gt/2, With E= -1 then r(t) =G(cosh(t) -1), and with E = +1 then r(t) = G(1-cos(t)). G is related to Newton's gravitational constant and = 64000. Write a program that will asks the user for the epsilon value from which the **curve** of r versus t will be produced within the given time interval.
The program should terminate if the user enters a non-valid value of E. HINT: The while loop should be controlled with 'or' logic for the three values of E. After the **curves ie plots for each case** are shown the program **should produce the value of r at the half way point (t=pi!)** with some appropriate comment. Use fprintf() for outputs. Run the program for all three epsilon cases, as well as, termination to be sure it is working. **USE WHILE loop and If –elseif else structure**

27(1 pt). EZPLOT! MATLAB has many commands which we will not touch on but from time to time it’s a good idea to explore. Another plotting routine function is ezplot. Explore this with >>help ezplot and look at this example: ezplot('5*y^4-3*y^3+2',[-8,8]);
Now plot with EZPLOT command and hand in the function: $\sin(7x)e^{-0.04x}$ range $-50< x<+50$
Be sure to put your name on the graph next to the equation heading, that is, by expanding the title to include your name and submit the graph only.