Problem 1.
The following is a recursive MATLAB function that sometimes prints `Hello` and (really) does nothing else other than calling itself recursively. If we call this function with `ps8hello(4)`, how many times do we have the line `Hello` printed? Explain.

```
function ps8hello(n)
  if (n==0)
     fprintf('Hello \n');
  else
     for kk=0:(n-1)
       p5hello(kk);
     end
  end
end
```

Problem 2.
You are given the recursive function `findme`. What is the value returned in `a` for `a=findme(91)` ?

```
function y=findme(x)
  if (x> 100)
    y=x-10;
    return;
  else
    y=findme(findme(x+11));
    return;
  end
end
```

Problem 3.
You are given the recursive function `exam2p4` from a past exam. What does it do? What is the value printed if we call it `exam2p4(10)`? Explain.

```
function s=exam2p4(n)
% n is a positive base-10 integer
%
if(n==0)
    s='0';
elseif(n==1)
    s='1';
else
    s=[exam2p4(floor(n/2)),exam2p4(mod(n,2))];
end

end
```

Problem 4.

```
% This is file p4.m
 function [y x] =  p4 ( x , y)
   a= 10 * x + 5 * y;
   b= x + y;
   x= b;
   y= a;
end
% This is the end of file  p4.m
```

The following code is executed in MATLAB's command window. What are the values of q, t?

```
>> [q t ] =  p4(10,5);
>> q

>> t
```

Problem 5.

With respect to Problem 1, we now execute the following MATLAB code. What are the values of q, t?

```
>> [q t ] = qz6(10,5);
>> [q t ] = qz6(t, 5);
>> q

>> t
```

Turn page for solutions

Representative Solutions to the listed problems

Problem 1.

`Hello` will get printed 8 times total. Let us see why. The `Hello` gets printed only when $n = 0$ i.e. the if statement of line 2 is `true`. Otherwise (if n is not 0), the loop of lines 5-7 will get executed that call `ps8hello` recursively n times with arguments `ps8hello(0) , ... , ps8hello(n-1)`. The indentation shows only the level of the recursive calls

```
  ps8hello(4)
%   1  2  3  4
    ps8hello(0)  %
    Hello                % 1st hello
    ps8hello(1)
       ps8hello(0) %
       Hello             % 2nd hello
    ps8hello(2)
       ps8hello(0)
       Hello             % 3rd hello
       ps8hello(1)
          ps8hello(0)
          Hello          % 4th
    ps8hello(3)
        ps8hello(0)
        Hello            % 5th
        ps8hello(1)
           ps8hello(0)
           Hello         %6th
        ps8hello(2)
           ps8hello(0)
           Hello         %7th
           ps8hello(1)
              ps8hello(0)
              Hello      %8th
```

Step A. Let us trace the execution of `ps8hello(4)`. Lines 4-6 result in issuing `ps8hello(0)`, `ps8hello(1)`, `ps8hello(2)`, `ps8hello(3)` respectively.
Step B0. A `ps8hello(0)` prints `Hello` and terminates.
Step B1. But then a `ps8hello(1)` will also issue a `ps8hello(0)` that by B0, prints `Hello` and terminates.
Step B2. A `ps8hello(2)` will call `ps8hello(0)` and a `ps8hello(1)`, i.e. by B0,B1, print `Hello` twice.
Step B3. And a `ps8hello(3)` will call all three `ps8hello(0)`, `ps8hello(1)`, `ps8hello(2)`, i.e. by B0,B1,B2 print `Hello` $1 + 1 + 2 = 4$ times.
Step C. Combining B0,B1,B2,B3 we observe that a total of eight `Hello` will be printed.
Generalization. If `ps8hello(n)` was issued, then `Hello` would be printed 2^{n-1} times!

In the trace below, a `ps8hello` function that is being called recursively from within another `ps8hello` function is indented to the right 6 positions.

```
>> ps8hello(4)
     ps8hello(0)                        ----->  Hello
     ps8hello(1)
          ps8hello(0)                   ----->  Hello
     ps8hello(2)
          ps8hello(0)                   ----->  Hello
          ps8hello(1)
               p5hello(0)               ----->  Hello
     ps8hello(3)
          ps8hello(0)                   ----->  Hello
          ps8hello(1)
               ps8hello(0)              ----->  Hello
          ps8hello(2)
               ps8hello(0)              ----->  Hello
               ps8hello(1)
                    ps8hello(0)         ----->  Hello
```

Problem 2.

We provide the recursive calls to show what is happening with the call `a=findme(91)`.

```
function y=findme(x)                    Line 1
  if (x> 100)                                2
    y=x-10;                                  3
    return;                                  4
  else                                       5
    y=findme(findme(x+11));                  6
    return;                                  7
  end                                        8
end
```

```
            findme(91)                          :Line 1
                 findme(findme(91+11))          :Line 6/7
                 findme(findme(102)  )          :Line 1
                 findme(     92       )         :Line 3/4
                    findme(findme(92+11))       :Line 6/7
                    findme(     93        )     :Line 3/4
                       findme(findme(93+11))    :Line 6/7
                       findme(     94       )         :Line 3/4
                          findme(findme(94+11))       :Line 6/7
                          findme(     95       )      :Line 3/4
                             findme(findme(95+11))    :Line 6/7
                             findme(     96       )   :Line 3/4
                                findme(findme(96+11)) :Line 6/7
                                findme(     97       )       :Line 3/4
                                   findme(findme(97+11))     :Line 6/7
                                   findme(     98       )    :Line 3/4
                                      findme(findme(98+11))  :Line 6/7
                                      findme(     99       ) :Line 3/4
                                         findme(findme(99+11))     :Line 6/7
                                         findme(     100      )    :Line 3/4
                                            findme(findme(100+11)) :Line 6/7
                                            findme(     101      ) :Line 3/4
                                                   y=101-10=91     :Line 3/4
                                                   return
```

```
Value returned is 91.
```

Problem 3. Let us examine carefully the code for this function. In the first line (after the percent comments) we check (i) whether n is equal to 0, and set s to 0, in the following `elseif` we check (ii) whether n is equal to 1 and set s to 1, and in all other cases we set s to an array (see the brackets) consisting of two parts, one involving `exam2p4(floor(n/2))` and the other involving `exam2p4(mod(n,2))`. Thus because of these two calls function `exam2p4` is a recursive function. The first two cases are the base cases.

Summarizing we have

Case 1. If `n == 0` then `s = '0'`.

Case 2. If `n == 1` then `s = '1'`.

Case 3. otherwise, then `s= [ exam2p4(floor(n/2)) , exam2p4(mod(n,2)) ]`

Cases 1 and 2 are the base cases of the recursive function. Case 3 is the interesting case to be further analyzed. The operations $s = [xy]$ forms a vector of x and y. If x, y are numbers then s is a row vector of two numbers. If x, y are two row vectors themselves, the s is a row vector with a number of elements equal to the sum of the elements of x, y. What if x, y are strings of characters such as `'0', '1'`? Then s is the concatenation of the strings x, y respectively!

What does `floor(n/2) , mod(n,2)` do for various values of n?

And then what do `exam2p4(floor(n/2)) , exam2p4(mod(n,2))` return?

```
 n    floor(n/2)    mod(n,2)
 1    0             1
 2    1             0            2 is  1 0
 3    1             1
 4    2             0
 5    2             1            5 is  2 1
10    5             0           10 is  5 0
```

It is obvious that `n = 2 * floor(n/2) + mod(n,2)`, i.e. `floor(n/2) , mod(n,2)` are the quotient and remainder of the division of n by 2. In the fourth column above we wrote for selected numbers its results `floor(n/2) , mod(n,2)` collected from columns 2 and 3. The we reverse the process, i.e. instead of writing 10 we write a `5 0`. Instead of writing a `5 0` we replace the 5 by its `2 1` representation and then the 2 by its `1 0` representation.

```
      10 is 5 0
since  5 is 2 1  we can write 10 as 5 0   or 10 as 2 1 0 as well
since  2 is 1 0  we can write 10 as 5 0   or 10 as 2 1 0 or furthermore 10 as  1 0 1 0
But 1 0 10 is the binary representation of 10 since 1010 = 1*2^3 + 1*2^1 = 8+2 = 10
```

So the call, `exam2p4(floor(n/2))` tells function `exam2p4` to do something on the quotient of the division of n by 2 and `exam2p4(mod(n,2))` tells function `exam2p4` to do something on the remainder of the division of n by 2. What it does is to find the binary representation of both in order to derive the binary representation of n.

Thus collecting all the pieces we can trace the execution of `exam2p4(10)` as follows

```
exam2p4(10) =
 Case 3 for s below
 s = [         exam2p4(5)                                    ,      exam2p4(0)   ];
       Case 3 for s below                                      Case 1 for s below
 s = [ s=[     exam2p4(2),                exam2p4(1)   ] , s=  '0'           ];
               Case 3                     Case 2             value returned is s
 s = [ s=[     s=[ exam2p4(1),exam2p4(0)], s= '1'       ] ,     '0'           ];
               Case 2         Case 1
 s = [ s=[     s=[ s= '1'    , s='0'    ],    '1'       ] ,     '0'           ];
                   value returned is s
 s = [ s=[     s=[    '1'    ,   '0'    ],    '1'       ] ,     '0'           ];
               value returned is s
 s = [ s=[       '10'                      ,    '1'       ] ,     '0'           ];
       value returned is s
 s = [ '101'                                                   ,     '0'           ];
 s = [ '1010'                                                                      ];
%  value returned is the value of string s
%  where  s = '1010';
```

Problem 4.

```
>> [q t ] =  p4(10,5);                    % [q t] =  p4(10,5) = <<< [ y x ] = [ 125 15]
>> q                                      %  q= 125

>> t                                      %  t= 15
```

Problem 5.

```
>> [q t ] =  p4(10,5);
>> [q t ] =  p4(t, 5);
>> q                                   % q=175

>> t                                   % t= 20
```