Assignment One

Rule:

1. Every subpart of assignment should be evaluated out of 10 Marks
2. Copy Case -10 marks on each subparts
3. a) Able to write the Ocaml function
   b) Able to execute the Ocaml function
   c) Time Complexity of Algorithm
   d) Testing for all cases of Inputs
   e) Indentation and Organization/Naming of functions
   f) Commenting in the program
   g) Judging the knowledge about the programming

Exercise 3: Write a function area for computing the area of a circle whose diameter is given.

Solution1:

```
let pi = 22.0 /. 7.0 ;;(* Defining PI *)
let area : float->float =
            function r-> pi *. r *. r;;
area 5.0;;(* Test Cases *)
area 3.0;;(* Test Cases *)
```

Solution2:

```
let pi = 22.0 /. 7.0 ;;(* Defining PI *)
let square : float->float =
      function x -> x *. x ;;
let area : float->float =
      function r-> pi *. square(r);;
```

* if PI is define locally(in side function), give more credit
* With a square function will be efficient

Exercise 5 Write a Caml function to determine the roots of a quadratic equation. Figure out a way to handle the case when the the roots are not real.

Solution1:

```
let desc : float*float*float->float =
function (a,b,c)->(b *. b) -. 4.0 *. a *. c ;;
let mdesc : float*float*float->float =
function (a,b,c)-> 4.0 *. a *. c -. (b *. b);;
let root: float*float*float->string =
 function (a,b,c) ->
   if ( desc(a,b,c) > 0.0 ) then
"root1=" ^ string_of_float(( ~-. b +. sqrt(desc(a,b,c) ) /. 2.0 *. a) )
^ "root2=" ^ string_of_float(( ~-. b -. sqrt(desc(a,b,c)) /. 2.0 *. a) )

  else
```

```
                  "root1=" ^ string_of_float((~-. b)/. 2.0 *. a)^ " + "
                  ^ string_of_float( sqrt( mdesc(a,b,c) ) /. 2.0 *. a) ^ " i "

                  ^ "root2=" ^ string_of_float((~-. b)/. 2.0 *. a)^ " - "
                  ^ string_of_float( sqrt( mdesc(a,b,c) ) /. 2.0 *. a) ^ " i " ;;
```

Solution2:

a)mdesc is not required

b) In place of else part output, they may have output "No Real Root"

c) If root is float*float*float->float*float, they can not handle string and output should be (0,0)

Solution3:

```
let desc : float*float*float->float =
function (a,b,c)->(b *. b) -. 4.0 *. a *. c ;;
let root: float*float*float->float*float =
      function (a,b,c)= if ( desc(a,b,c) > 0.0 ) then
      ( ( ~-. b +. sqrt(desc(a,b,c) ) /. 2.0 *. a),
      ( ~-. b -. sqrt(desc(a,b,c)) /. 2.0 *. a))
      else
      (0.0,0.0);;
```

Exercise 6 Let d be an integer and m be a string. Write a Caml function that returns true iff d and m form a valid date (assume non-leap year). For example 31 April is not a valid date.

Soution:

```
let validdate: int*string -> bool =
            function (date, month)->
                  if( date <1 && date >31 ) then
                  false
                        else
                              if (
                                    month=="Jan" ||
                                    (month=="Feb" && date<29) ||
                                    month=="Mar" ||
                                    (month=="Apr" && date<31) ||
                                    month=="May" ||
                                    (month=="Jun" && date<31) ||
                                    month=="Jul" ||
                                    month=="Aug" ||
                                    (month=="Sep" && date<31) ||
                                    month=="Oct" ||
                                    (month=="Nov" && date<31) ||
                                    month=="Dec"
                                  )
                               then true
                               else false ;;
```

1.Checking for numeric date should before checking for string

2. Other Method of matching also can be done.

Exercise 8 The greatest-common-divisor (gcd) of two non-negative integers m, n is known to satisfy the identity gcd(m, n) = gcd(m, n + m). Prove it. Then use it to give a recursive definition of gcd of two (non-negative) integers in Caml.

Solution:

```
gcd(m, n) = gcd(m, n + m) is same as gcd(m, n + m) = gcd(m, n)
Adding -m to 2nd term
                        gcd(m, n+m-m ) = gcd(m,n-m)
Again it is same as      gcd(m, n     ) = gcd(m,n-m);
```

```
let rec gcd : int * int -> int =
function (m,n) ->
            if (m==n)
                  then m
            else if(m>n)
                  then gcd(n,m-n)
                  else
                       gcd(m,n-m) ;;
```

Solution1: for speeding up the convergence, mod operator can be used..

```
let rec gcd : int * int -> int =
      function (m,n) ->
            if (m==n)
                  then m
            else if(m>n)
                  then gcd(n,m mod n)
                  else
                       gcd(m,n mod m) ;;
```

Exercise 9 Using the observation that ${x^k}^2= x^(2k)$, give an alternate recursive definition of the function power(x, n) that computes x^n and write a corresponding Caml program.

Hint : If n is odd then n-1 is even !

```
let rec power: int*int->int =
function (x,n)->
      if (n==0) then 1
      else if ((n mod 2)==0)
            then power(x, n/2)*power(x,n/2)
            else power(x, (n-1)/2)*power(x,(n-1)/2)*x;;
```

* Credit for Handling Base Case
* Extra Credit for Handling Negative and Negative Power Case
• Use of Square function may degrade the performance

Find out the value of max_int without using the the constant max_int

Solution1:

Assuming that the max_int number is a general number it may not be in the format (2^n)-1.

Use of binary search ..to find max int and it have two phase, one is multiplication phase and another is addition phase.

```
example
            Assume 123 is maximum ..
   start with 1..
    (Multiplication Phase)
   search procedure start searching at 2 and go for 4,8,16,32,64,...
   after then we encounter a negative..
    (Addition Phase)
           it will go for 96(64+32), 112(96+16), 120(112+8),
           if we add 4 to 120 again  we encounter a negative  so
            we will add 2 instead of 4..so go for 122(120+2) and then go
           for (122+1)..and 123 is max_int
   Start searching from zero

       let rec findmax_addphase int*int ->int
              function x,y -> if(y=0) then x
                              else
                                 if ( x+ (y/2) > 0 )
                                    then findmax_addphase(x+y,y/2)
                                 else
                                       findmax_addphase(x,y/2);;

       let rec findmax_mulphase:int->int =
             function x-> if( x*2 < 0 ) then findmax_addphase(x,x);
                   else findmax_mulphase(x*2);;

       findmax_mulphase(1);; (* can n't be zero *)
```

Solution2:

People may think of
1+1+1+......upto a negative result,
Instead of adding one pleople tried to add 1000 or 100000
Will Take Time is a O(n) algorithm ..

Solution3:

```
Assuming max_int is (2^n)-1 form Simple and Solution is
              value of max_int = 1073741823  = 1073741824-1 =2^(30)-1
              It is Max_int in a 32 Bit Machine, One SignBit, One Extra Bit
              Logic Behind this is 1073741823+(1) is negative number
              2*2*2......can go upto 2^(29) after that you will get negative number

              Previous power function can be used efficiently.
                  2^(30)-1 = 2^(29)-1+2^(29)
                  Solution1: (Its the Best Solution)
                  let rec findmax2:int->int =
                        function x-> if( x*2 < 0 ) then x-1+x
                              else findmax2(x*2);;
                        findmax(1);; (* can n't be zero *)
```

Solution4:

```
       (* Start from 0 *)
       let rec findmax1:int->int =
             function x->
                   if(power(2,x)>0) then findmax1(x+1)
```

```
             else power(2,x-1)-1+power(2,x-1);;
findmax1(0);;
(* Wrting a function which make bound the input to 0 only*)
let rec findmax:int->int =
function x-> if(x<0 && x>0) then findmax(0)
else findmax1(0);;
```