Lab 8 solutions
COL 100

```
(*----     Module begins    ----*)
module Geometry = struct

(* Defining the type for the figures like point, Line, Circle, Polynomial
*)
type shape =
      | Point of (int*int)
      | Line of ((int*int)*(int*int))
      | Circle of ((int*int)*int)
      | Poly of (int*int) list

(* Declaring separate lists for each type of figures *)
let point_list = ref []
let line_list = ref []
let circle_list = ref []
let poly_list = ref []

let rec sep s =
      if String.contains s (' ') then
            (String.sub s 0 (String.index s ' '))::sep (String.sub s
((String.index s ' ')+1) ((String.length s)-((String.index s ' ')+1)))
      else [s]

let rec adder n n1 =
      (fst n + fst n1, snd n + snd n1)

let find_area n =
      match n with
      | Point (x,y) -> 0.0
      | Circle ((x,y),z) -> (3.14159265 *. float_of_int z *. float_of_int
z)
      | Line ((x,y),(w,z)) -> 0.0
      | _ -> 1.0

let find_perimeter n =
      match n with
      | Point (x,y) -> 0.0
      | Circle ((x,y),z) -> (2.0 *. 3.14159265 *. float_of_int z)
      | Line ((x,y),(w,z)) -> sqrt (((float_of_int x -. float_of_int
w)**2.0) +. ((float_of_int y -. float_of_int z)**2.0))
      | _ -> 1.0

let find_perimeter_poly inp =
      match inp with
      |  Poly n -> let temp = ref (List.hd n) in
            let rec helper n =
                  match n with
                  | [x] -> find_perimeter (Line ((x),(!temp)))
                  | hd::hd2::tl -> find_perimeter (Line ((hd),(hd2))) +.
helper (hd2::tl)
            in helper n

let find_distance n =
      let ((x,y),(w,z)) = n in
            sqrt (((x -. w)**2.0) +. ((y -. z)**2.0))

let give_centroid n =
```

```
      match n with
      | Point (x,y) -> (float_of_int x,float_of_int y)
      | Circle ((x,y),z) -> (float_of_int x,float_of_int y)
      | Line ((x,y),(w,z)) ->
((float_of_int(x+w))/.2.0,(float_of_int(y+z))/.2.0)
      | Poly l -> let (a,b) = (List.fold_left adder (0,0) l) in
(((float_of_int a)/. (float_of_int (List.length l))),((float_of_int b)/.
(float_of_int (List.length l) )))

(*
num denotes figure number, n denotes figure
It adds the figure n in the list of its type
*)
let add_to_lists num n =
      match n with
      | Point (x,y) -> point_list := (num, n)::(!point_list)
      | Circle ((x,y),z) -> circle_list := (num, n)::(!circle_list)
      | Line ((x,y),(w,z)) -> line_list := (num, n)::(!line_list)
      | _ -> poly_list := (num, n)::(!poly_list)


(*
l - list of some figure
num - figure number
function returns true if the fig no. num exists in the list l, false
otherwise
*)
let rec check_list l num =
      match l with
      | (a, b)::tl -> if num = a then true else check_list tl num
      | [] -> false

(*
Input: Figure List l and figure num
Output: Figure num from list l
*)
let rec give_list l num =
      match l with
      | (a, b)::tl -> if num = a then b else give_list tl num

let count_bracket inp =
      let n = ref 0 in
            for i = 0 to ((String.length inp)-1) do
                  if (inp.[i] = '(') then n := !n+1
      done;
      !n

let count_comma inp =
      let n = ref 0 in
            for i = 0 to ((String.length inp)-1) do
                  if (inp.[i] = ',') then n := !n+1
      done;
      !n

let rec give_to_point inp =
      let [x;y] = inp in
            let shape_number = int_of_char (x.[0]) in
                  let a = int_of_string (String.sub y 1 ((String.index y
',')-1)) in
```

```
                        let b = int_of_string (String.sub y ((String.index
y ',')+1) ((String.index y ')')-((String.index y ',')+1))) in
                              add_to_lists shape_number (Point (a,b))

let rec give_to_circle inp =
      let [x;y;z] = inp in
            let shape_number = int_of_char (x.[0]) in
                  let a = int_of_string (String.sub y 1 ((String.index y
',')-1)) in
                        let b = int_of_string (String.sub y ((String.index
y ',')+1) ((String.index y ')')-((String.index y ',')+1))) in
                              let c = int_of_string z in
                                    add_to_lists shape_number (Circle
((a,b),c))

let rec class_to_poly inp =
      match inp with
      | hd::tl -> begin
                  let a = int_of_string (String.sub hd 1 ((String.index hd
',')-1)) in
                        let b = int_of_string (String.sub hd
((String.index hd ',')+1) ((String.index hd ')')-((String.index hd
',')+1))) in
                              (a,b)::class_to_poly (tl)
            end
      | [] -> []

let rec give_to_poly inp =
      let x = List.hd inp in
            let shape_number = int_of_char (x.[0]) in
                  add_to_lists shape_number (Poly (class_to_poly (List.tl
inp)))

let rec give_to_line inp =
      let [x;y;z] = inp in
            let shape_number = int_of_char (x.[0]) in
                  let a = int_of_string (String.sub y 1 ((String.index y
',')-1)) in
                        let b = int_of_string (String.sub y ((String.index
y ',')+1) ((String.index y ')')-((String.index y ',')+1))) in
                              let c = int_of_string (String.sub z 1
((String.index z ',')-1)) in
                                    let d =  int_of_string (String.sub z
((String.index z ',')+1) ((String.index z ')')-((String.index z ',')+1)))
in
                                    add_to_lists shape_number (Line
((a,b),(c,d)))

let classify inp =
      if (count_bracket inp = 1 && count_comma inp = 1) then
give_to_point (sep inp)
      else if (count_bracket inp = 1 && count_comma inp = 2) then
give_to_circle (sep inp)
      else if (count_bracket inp = 2 && count_comma inp = 3) then
give_to_line (sep inp)
      else give_to_poly (sep inp)

let give_distance inp =
      let n1 = int_of_char (inp.[2]) in
            let n2 = int_of_char (inp.[4]) in
```

```
                if (check_list !point_list n1 = true) then begin
                      if (check_list !point_list n2 = true) then
find_distance ( ((give_centroid (give_list !point_list
n1)),(give_centroid (give_list !point_list n2))))
                      else if (check_list !circle_list n2 = true) then
find_distance ( ((give_centroid (give_list !point_list
n1)),(give_centroid (give_list !circle_list n2))))
                      else if (check_list !line_list n2 = true ) then
find_distance  ( ((give_centroid (give_list !point_list
n1)),(give_centroid (give_list !line_list n2))))
                      else find_distance  ( ((give_centroid (give_list
!point_list n1)),(give_centroid (give_list !poly_list n2)))) end
           else if (check_list !circle_list n1 = true) then begin
                      if (check_list !point_list n2 = true) then
find_distance ( ((give_centroid (give_list !circle_list
n1)),(give_centroid (give_list !point_list n2))))
                      else if (check_list !circle_list n2 = true) then
find_distance ( ((give_centroid (give_list !circle_list
n1)),(give_centroid (give_list !circle_list n2))))
                      else if (check_list !line_list n2 = true ) then
find_distance ( ((give_centroid (give_list !circle_list
n1)),(give_centroid (give_list !line_list n2))))
                      else find_distance ( ((give_centroid (give_list
!circle_list n1)),(give_centroid (give_list !poly_list n2)))) end
           else if (check_list !line_list n1 = true ) then begin
                      if (check_list !point_list n2 = true) then
find_distance ( ((give_centroid (give_list !line_list n1)),(give_centroid
(give_list !point_list n2))))
                      else if (check_list !circle_list n2 = true) then
find_distance ( ((give_centroid (give_list !line_list n1)),(give_centroid
(give_list !circle_list n2))))
                      else if (check_list !line_list n2 = true ) then
find_distance  ( ((give_centroid (give_list !line_list
n1)),(give_centroid (give_list !line_list n2))))
                      else find_distance  ( ((give_centroid (give_list
!line_list n1)),(give_centroid (give_list !poly_list n2)))) end
           else  begin
                      if (check_list !point_list n2 = true) then
find_distance ( ((give_centroid (give_list !poly_list n1)),(give_centroid
(give_list !point_list n2))))
                      else if (check_list !circle_list n2 = true) then
find_distance ( ((give_centroid (give_list !poly_list n1)),(give_centroid
(give_list !circle_list n2))))
                      else if (check_list !line_list n2 = true ) then
find_distance  ( ((give_centroid (give_list !poly_list
n1)),(give_centroid (give_list !line_list n2))))
                      else find_distance  ( ((give_centroid (give_list
!poly_list n1)),(give_centroid (give_list !poly_list n2)))) end

let using l =
     let b = snd (List.hd l) in
     let rec helper inp =
     match inp with
     | [(i,j)] -> i*b
     | (x,y)::(w,z)::tl ->  (x*z) + (helper ((w,z)::tl))
in helper l

let using2 l =
     let a = fst (List.hd l) in
     let rec helper inp =
```

```
      match inp with
      | [(i,j)] -> j*a
      | (x,y)::(w,z)::tl ->  (y*w) + (helper ((w,z)::tl))
in helper l

let find_area_poly l =
      match l with
      | Poly n -> abs_float (((float_of_int (using n)) -. (float_of_int
(using2 n))) /. 2.0)

let give_perimeter inp =
      let n = int_of_char (inp.[2]) in
            if (check_list !circle_list n = true) then find_perimeter
(give_list !circle_list n)
            else find_perimeter_poly (give_list !poly_list n)

let give_area inp =
      let n = int_of_char (inp.[2]) in
            if (check_list !circle_list n = true) then find_area
(give_list !circle_list n)
            else find_area_poly (give_list !poly_list n)


(* n stores the figure number *)
let give_shape inp =
      let n = int_of_char (inp.[2]) in
            if (check_list !point_list n = true) then "P"
            else if (check_list !circle_list n = true) then "C"
            else if (check_list !line_list n = true) then "L"
            else match (give_list !poly_list n) with Poly l -> "PLG
"^string_of_int (List.length l)

end
(*----     Module ends----*)

(* Reading each line of input file
Base case - If it an action command, then call the concerned function and
print the output
Else - If it is a figure, then call the function that classifies it as
point, circle, line or polygon and add it to the list of its type
*)
open Geometry
let () =
      let rec read_input () =
            try let file_input = (read_line()) in begin
                  if file_input.[0] = 'P' then print_string
(string_of_float (give_perimeter file_input)^"\n")
                  else if file_input.[0] = 'A' then print_string
(string_of_float (give_area file_input)^"\n")
                  else if file_input.[0] = 'T' then print_string
(give_shape file_input^"\n")
                  else if file_input.[0] = 'D' then print_string
(string_of_float (give_distance file_input)^"\n")
                  else classify file_input end ; read_input()
            with e ->
                  print_string ""
            in read_input()
```