```
exception Not_valid_oper;;
let isValidInfix stng =
   let opens = Stack.create() in
   let rec aux stng  n =
      if n >= (String.length stng) then opens
      else
      let s = String.get stng n in
      match s with
      '(' -> (Stack.push s opens ; aux stng (n+1))
      |')' -> let out_brack = Stack.pop opens in
            aux stng (n+1)
      | _ -> aux stng (n+1) in
   try
      Stack.length (aux stng 0) = 0
   with Stack.Empty -> false;;



let preced op =
   match op with
   "-" -> 1
   |"+" -> 1
   |"*" -> 2
   |"/" -> 2
   |"^" -> 3
            |"?" -> 3
   |"log" -> 4
            | _ -> raise(Not_valid_oper)
   ;;

let if_oper op stck =
   let rec aux op stck s =
      if Stack.is_empty stck then (Stack.push op stck; s)
      else
                              let top = Stack.top stck in
                              if top = "(" then (Stack.push op stck; s)
                              else if preced top >= preced op then (s := !s ^ " " ^ (Stack.pop stck); aux op stck s)
                              else
                                 (Stack.push op stck; s) in
                                       let s = ref "" in
   !(aux op stck s);;

let rec split_on_char c s =
            try
                        let char_index = String.index s c in
                        let top_word, rest = String.sub s 0 char_index, String.sub s (char_index+1) ((String.length s)-
(char_index+1)) in
                              if not(rest = "") then top_word :: (split_on_char c rest)
                              else [top_word]
            with Not_found -> [s];;

let operators = ["/";"*";"+";"-";"^";"log";"?"];;

let in_to_post s =
            let help_stack = Stack.create() in
            let rec aux s_list reference stck =
                        match s_list with
                        [] -> reference
                        |h::r -> if h = "(" then(Stack.push h stck; aux r reference stck )
                                          else if not((List.mem h operators) || h = ")") then (reference := !reference ^ " " ^ h ;
aux r reference stck )
                                          else if h = ")" then
```

```
                                        (while not(Stack.top stck = "(" || Stack.is_empty stck) do
                                                reference := !reference ^ " " ^ (Stack.pop stck)
                                        done;
                                        if Stack.is_empty stck then aux r reference stck
                                        else
                                                let throw_out = Stack.pop stck in
                                                        aux r reference stck)
                                else
                                        (reference := !reference ^ (if_oper h stck);
                                        aux r reference stck) in
        let new_ref = aux (split_on_char ' ' s) (ref "") help_stack in
                (while not(Stack.is_empty help_stack) do
                        if Stack.top help_stack = "(" then let throw_out = Stack.pop help_stack in new_ref := !
new_ref
                        else
                        new_ref := !new_ref ^ " " ^ (Stack.pop help_stack)
                done;
                String.trim (!new_ref));;

let rec take_in ic =
        try
                let line = input_line ic in
                        line :: take_in ic
        with End_of_file -> [];;

let rec print_out expsns =
        match expsns with
        [] -> print_string ""
        |h::r -> if isValidInfix h then (print_string((String.trim(in_to_post h)) ^ "\n");
                                                print_out r)
                 else
                        (print_string "not valid" ; print_string "\n";
                        print_out r);;

let () =
        let expsns = take_in stdin in
                print_out expsns;;
-------------------------------------------------------------------------------------------------------------------------------
let truncate d_places f_string =
        if not(String.contains f_string '.') then f_string
        else if (String.index f_string '.' = (String.length f_string)-1) then String.sub f_string 0 ((String.length f_string) -
1)
        else
        let len = (String.index f_string '.') + 1 + d_places in
                if String.length f_string > len then
                        String.sub f_string 0 len
                else f_string;;

let isInt s =
        not(String.contains s '.' || (String.index s '.' = (String.length s)-1));;

let make_float s =
        if isInt s then string_of_float(float_of_int(int_of_string s))
        else
                s;;

let isValidInfix stng =
   let opens = Stack.create() in
   let rec aux stng n =
      if n >= (String.length stng) then opens
      else
      let s = String.get stng n in
```

```
      match s with
      '(' -> (Stack.push s opens ; aux stng (n+1))
      |')' -> (ignore(Stack.pop opens);
            aux stng (n+1))
      | _ -> aux stng (n+1) in
   try
      Stack.length (aux stng 0) = 0
   with Stack.Empty -> false;;

let precedence op =
   match op with
   "-" -> 1
   |"+" -> 1
   |"*" -> 2
   |"/" -> 2
   |"^" -> 3
           |"?" -> 3
   |"log" -> 4
           | _ -> raise (Failure "Not_valid_operator")
   ;;

let if_operator op stck =
   let rec aux op stck s =
      if Stack.is_empty stck then (Stack.push op stck; s)
      else
                              let top = Stack.top stck in
                              if top = "(" then (Stack.push op stck; s)
                              else if precedence top >= precedence op then (s := !s ^ " " ^ (Stack.pop stck); aux op stck s)
                              else
                                (Stack.push op stck; s) in
                                      let s = ref "" in
   !(aux op stck s);;

let rec split_on_char c s =
           try
                   let char_index = String.index s c in
                   let top_word, rest = String.sub s 0 char_index, String.sub s (char_index+1) ((String.length s)-
(char_index+1)) in
                              if not(rest = "") then top_word :: (split_on_char c rest)
                              else [top_word]
           with Not_found -> [s];;

let operators = ["/";"*";"+";"-";"^";"log";"?"];;

let in_to_post s =
           let help_stack = Stack.create() in
           let rec aux s_list reference stck =
                   match s_list with
                   [] -> reference
                   |h::r -> if h = "(" then(Stack.push h stck; aux r reference stck )
                                      else if not((List.mem h operators) || h = ")") then (reference := !reference ^ " " ^ h ;
aux r reference stck )
                                      else if h = ")" then
                                              (while not(Stack.top stck = "(" || Stack.is_empty stck) do
                                                      reference := !reference ^ " " ^ (Stack.pop stck)
                                              done;
                                              if Stack.is_empty stck then aux r reference stck
                                              else
                                                      (ignore(Stack.pop stck);
                                                              aux r reference stck))
                                      else
                                              (reference := !reference ^ (if_operator h stck);
```

```
                                        aux r reference stck) in
        let new_ref = aux (split_on_char ' ' s) (ref "") help_stack in
                (while not(Stack.is_empty help_stack) do
                        if Stack.top help_stack = "(" then (ignore(Stack.pop help_stack); new_ref := !new_ref)
                        else
                        new_ref := !new_ref ^ " " ^ (Stack.pop help_stack)
                done;
                String.trim (!new_ref));;

let evaluate_postfix s =
        try
        (let arr, stck = Array.of_list(split_on_char ' ' s), Stack.create() in
                for i = 0 to ((Array.length arr)-1) do
                                let elem = arr.(i) in
                                (if (List.mem elem operators) then (match elem with
                                        "-" -> let a, b = Stack.pop stck, Stack.pop stck in
                                                        Stack.push (string_of_float((float_of_string a) -.
(float_of_string b))) stck
                                        |"+" -> let a, b = Stack.pop stck, Stack.pop stck in
                                                        Stack.push (string_of_float((float_of_string a) +.
(float_of_string b))) stck
                                        |"*" -> let a, b = Stack.pop stck, Stack.pop stck in
                                                        Stack.push (string_of_float((float_of_string a) *.
(float_of_string b))) stck
                                        |"/" -> let a, b = Stack.pop stck, Stack.pop stck in
                                                        Stack.push (string_of_float((float_of_string a) /.
(float_of_string b))) stck
                                        |"^" -> let a, b = Stack.pop stck, Stack.pop stck in
                                                        Stack.push (string_of_float((float_of_string a) **
(float_of_string b))) stck
                                        |"?" -> let a, b = Stack.pop stck, Stack.pop stck in
                                                        Stack.push (string_of_float((float_of_string a) **
(float_of_string b))) stck
                                        |"log" -> let a = Stack.pop stck in
                                                        Stack.push (string_of_float(log (float_of_string a))) stck)
                                else
                                        (Stack.push elem stck))

                done;
                (let len = Stack.length stck in
                        if len = 0 then print_string ""
                        else if len = 1 then print_string (truncate 4 (Stack.pop stck))
                        else
                                print_string "not valid"))
        with e -> print_string "not_valid";;

let rec take_in ic =
        try
                let l = (input_line ic) in
                        l :: (take_in ic)
        with End_of_file -> [];;

let rec print_out expsns =
        match expsns with
        [] -> print_string ""
        |h::r -> if isValidInfix h then (evaluate_postfix (in_to_post h); print_string "\n";
                                                                        print_out r)
                        else
                                (print_string "not valid" ; print_string "\n";
                                print_out r);;

let () =
```

```
        let expsns = take_in stdin in
                print_out expsns;;
```

```
module BigNumber = struct
let rec remove_zero a =
if(a = "0") then a
else
if (a.[0] = '0') then remove_zero (String.sub a 1 (String.length(a)-1))
else a ;;
(* This function makes both a and b of equal length by appending the necessary amount of 0s in the beginning*)
let equal_length a b =
if (String.length a < String.length b) then ((String.make (String.length b-String.length a) '0')^a,b)
else (a,(String.make (String.length a-String.length b) '0')^b);;
(* This function checks if the number represented by string a is less than the number represented by string b *)
let is_less a b =
let new_A, new_B = equal_length a b in
(new_A < new_B);;
(*Addition code starts*)
(* This is the main function to add a and b in a recursive manner*)
let rec add_main a b carry sum =
let ones_a = (int_of_char a.[String.length(a)-1]) - 48 in
let ones_b = (int_of_char b.[String.length(b)-1]) - 48 in
let new_carry = (ones_a + ones_b + carry)/10 in
let curr_sum_digit = string_of_int ((ones_a + ones_b + carry) mod 10) in
if (String.length a = 1) then
(string_of_int new_carry) ^ curr_sum_digit ^ sum
else add_main (String.sub a 0 ((String.length a)-1)) (String.sub b 0 ((String.length b)-1)) new_carry
(curr_sum_digit^sum) ;;
(* This function just preprocesses the numbers a and b and then adds them*)
let add a b =
let (new_a, new_b) = equal_length a b in
let ans = add_main new_a new_b 0 "" in
remove_zero ans;;
(* This function adds a series of large numbers*)
let add_series l = List.fold_left add "" l;;
(*Addition code ends*)
(*Subtraction code starts*)
(* This is the main function to subtract b from a in a recursive manner*)
let rec sub_main a b borrow diff =
let ones_a = (int_of_char a.[String.length(a)-1]) - 48 in
let ones_b = (int_of_char b.[String.length(b)-1]) - 48 in
let temp = (ones_a - ones_b - borrow) in
let new_borrow = if(temp >= 0) then 0 else 1 in
let curr_diff_digit = if (temp >= 0) then string_of_int (temp)
else string_of_int (temp+10)
in
if (String.length a = 1) then
curr_diff_digit ^ diff
else sub_main (String.sub a 0 ((String.length a)-1)) (String.sub b 0 ((String.length b)-1)) new_borrow
(curr_diff_digit^diff) ;;
(* This function just preprocesses the numbers a and b and then subtracts them*)
1
let subtract a b =
let (new_a, new_b) = equal_length a b in
let ans = sub_main new_a new_b 0 "" in
remove_zero ans;;
(*Subtraction code ends*)
(* Multiplication code starts*)
(* This function multiplies a large number with a single digit *)
let rec mult_single a b carry mult =
let ones_a = (int_of_char a.[String.length(a)-1]) - 48 in
let ones_b = (int_of_char b) - 48 in
```

```
let new_carry = (ones_a*ones_b + carry)/10 in
let curr_mult_digit = string_of_int ((ones_a*ones_b + carry) mod 10) in
if (String.length a = 1) then
(string_of_int new_carry) ^ curr_mult_digit ^ mult
else mult_single (String.sub a 0 ((String.length a)-1)) b new_carry (curr_mult_digit^mult) ;;
(* This is the main function to add a and b in a recursive manner*)
let rec mult_main a b part_sum =
let partial_mult = remove_zero (mult_single a (b.[0]) 0 "") in
if (String.length b = 1) then add partial_mult (part_sum^"0")
else
mult_main a (String.sub b 1 ((String.length b)-1)) (add partial_mult (part_sum^"0")) ;;
(* This function just preprocesses the numbers a and b and then adds them*)
let multiply a b =
let ans = mult_main a b "" in
remove_zero ans;;
(* This function adds a series of large numbers*)
let mult_series l = List.fold_left multiply "1" l;;
(* Multiplication code ends*)
(* Division code starts*)
(* This function is used to find the quotient when a is divided by b*)
let rec div a b times =
if(is_less a b) then (string_of_int(times),a)
else div (subtract a b) b (times+1) ;;
(* This function is the main recursive function to divide a by b in a recursive long division manner*)
let rec divide_main a b quotient index =
if (index >= String.length a) then quotient
else
if (is_less (String.sub a 0 (index+1)) b) then divide_main a b (quotient^"0") (index+1)
else
let (curr_div_digit, remainder) = div (String.sub a 0 (index+1)) b 0 in
if(remainder = "0") then
let new_a = (String.sub a (index+1) (String.length(a) - index-1)) in
divide_main (new_a) b (quotient^curr_div_digit) (0)
else
let new_a = remainder ^ (String.sub a (index+1) (String.length(a) - index-1)) in
divide_main (new_a) b (quotient^curr_div_digit) (String.length(remainder)) ;;
(* It just handles the boundary cases for division and then makes the call to the divide_main function*)
let divide a b = if (is_less a b) then "0"
else if (b = "0") then "NAN"
else remove_zero (divide_main a b "" 0) ;;
(* Division code ends*)
end;;


let truncate d_places f_string =
        let len = (String.index f_string '.') + 1 + d_places in
                if String.length f_string > len then
                        String.sub f_string 0 len
                else f_string;;

let isInt s =
        not(String.contains s '.');;

let make_float s =
        if isInt s then string_of_float(float_of_int(int_of_string s))
        else
                s;;

let isValidInfix stng =
   let opens = Stack.create() in
   let rec aux stng n =
      if n >= (String.length stng) then opens
      else
```

```
      let s = String.get stng n in
      match s with
      '(' -> (Stack.push s opens ; aux stng (n+1))
      |')' -> (ignore(Stack.pop opens);
            aux stng (n+1))
      | _ -> aux stng (n+1) in
   try
      Stack.length (aux stng 0) = 0
   with Stack.Empty -> false;;

let precedence op =
   match op with
   "-" -> 1
   |"+" -> 1
   |"*" -> 2
   |"/" -> 2
         | _ -> raise (Failure "Not_valid_operator")
   ;;

let if_operator op stck =
   let rec aux op stck s =
      if Stack.is_empty stck then (Stack.push op stck; s)
      else
                              let top = Stack.top stck in
                              if top = "(" then (Stack.push op stck; s)
                              else if precedence top >= precedence op then (s := !s ^ " " ^ (Stack.pop stck); aux op stck s)
                              else
                                 (Stack.push op stck; s) in
                                       let s = ref "" in
   !(aux op stck s);;

let rec split_on_char c s =
         try
                  let char_index = String.index s c in
                  let top_word, rest = String.sub s 0 char_index, String.sub s (char_index+1) ((String.length s)-
(char_index+1)) in
                              if not(rest = "") then top_word :: (split_on_char c rest)
                              else [top_word]
         with Not_found -> [s];;

let operators = ["/";"*";"+";"-"];;

let in_to_post s =
         let help_stack = Stack.create() in
         let rec aux s_list reference stck =
                  match s_list with
                  [] -> reference
                  |h::r -> if h = "(" then(Stack.push h stck; aux r reference stck )
                                    else if not((List.mem h operators) || h = ")") then (reference := !reference ^ " " ^ h ;
aux r reference stck )
                                    else if h = ")" then
                                             (while not(Stack.top stck = "(" || Stack.is_empty stck) do
                                                      reference := !reference ^ " " ^ (Stack.pop stck)
                                             done;
                                             if Stack.is_empty stck then aux r reference stck
                                             else
                                                      (ignore(Stack.pop stck);
                                                               aux r reference stck))
                                    else
                                             (reference := !reference ^ (if_operator h stck);
                                             aux r reference stck) in
         let new_ref = aux (split_on_char ' ' s) (ref "") help_stack in
                  (while not(Stack.is_empty help_stack) do
```

```
                        if Stack.top help_stack = "(" then (ignore(Stack.pop help_stack); new_ref := !new_ref)
                        else
                        new_ref := !new_ref ^ " " ^ (Stack.pop help_stack)
                done;
                String.trim (!new_ref));;

let evaluate_postfix s =
        try
         (let arr, stck = Array.of_list(split_on_char ' ' s), Stack.create() in
                for i = 0 to ((Array.length arr)-1) do
                                let elem = arr.(i) in
                                (if not(List.mem elem operators) then (Stack.push elem stck)
                                else
                                        (match elem with
                                        "-" -> let a, b = Stack.pop stck, Stack.pop stck in
                                                        Stack.push (BigNumber.subtract a b) stck
                                        |"+" -> let a, b = Stack.pop stck, Stack.pop stck in
                                                        Stack.push (BigNumber.add a b) stck
                                        |"*" -> let a, b = Stack.pop stck, Stack.pop stck in
                                                        Stack.push (BigNumber.multiply a b) stck
                                        |"/" -> let a, b = Stack.pop stck, Stack.pop stck in
                                                        Stack.push (BigNumber.divide a b) stck
                                        | _ -> raise (Failure "not_valid_operator")))
                done;
                (let len = Stack.length stck in
                        if len = 0 then print_string ""
                        else if len = 1 then print_string (Stack.pop stck)
                        else
                                print_string "not valid"))
        with e -> print_string "not valid";;

let rec take_in ic =
        try
                let l = (input_line ic) in
                        l :: (take_in ic)
        with End_of_file -> [];;

let rec print_out expsns =
        match expsns with
        [] -> print_string ""
        |h::r -> if isValidInfix h then (evaluate_postfix (in_to_post h); print_string "\n";
                                                                                print_out r)
                         else
                                (print_string "not valid" ; print_string "\n";
                                print_out r);;

let () =
        let expsns = take_in stdin in
                print_out expsns;;
```