```
          open String;;

          module BigNumber = struct

          (* This function removes any leading 0s in the string a*)
          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

          (* 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))


          (* This function just preprocesses the numbers a and b and then subtracts them*)
```

```
    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;;

(* This function splits the string s and returns a list of strings*)
let rec parse s word t =
        if(String.length(s) = 0) then t@[word]
        else if(s.[0] = ' ') then parse (String.sub s 1 (String.length(s)-1)) "" (t@[word])
        else parse (String.sub s 1 (String.length(s)-1)) (word ^ (String.make 1 s.[0])) t ;;

(* This function evaluates any expression given in the form of a string*)
let evaluate s =
        let arg_list = parse s "" [] in
                match arg_list with
                | "ADD" :: t -> BigNumber.add_series t
                | "SUB" :: t -> BigNumber.subtract (List.nth t 0) (List.nth t 1)
                | "MULT" :: t -> BigNumber.mult_series t
                | "DIV" :: t -> BigNumber.divide (List.nth t 0) (List.nth t 1)
                | _ -> "";;


open BigNumber;;
open Printf;;

let () =
        let output_file = open_out(read_line()) in
        let rec read_input () =
        (* try ... with used for exception handling*)
        try
                let s = read_line() in
                let ans = evaluate s in
                fprintf output_file "%s\n" (ans);
                read_input ()
        with e -> close_out (output_file) in
        read_input()
(* close_out (output_file);; *)
```