```
      open Input
      let in_file=Sys.argv.(1)
      let out_file = Sys.argv.(2)
      type doc_info =
            {
                  doc_id : int;
                  freq : int;
                  positions : int list;
                  tf : float;
            }

      type word_info =
            {
                  word : string;
                  idf : float;
                  docs : doc_info list;
            }


      (*convert a string into list of (word,pos) *)
      let convert_to_word_list s =
            let convert x =
                  if(x == ',') then ' '
                  else if(x == '.') then ' '
                  else if(x == '!') then ' '
                  else x
            in
            let s1 = String.map convert s in
            let add_string l s pos=
                  if(s = "") then l
                  else l@[(String.lowercase s,pos)]
            in
            let rec parse s word t =
                  if(String.length s == 0) then
                        add_string t word (List.length t)
                  else if(s.[0] = ' ' && word = "")
                        then parse (String.sub s 1 (String.length(s)-1))
"" t
                  else if(s.[0] = ' ')
                        then parse (String.sub s 1 (String.length(s)-1))
"" (add_string t word (List.length t))
                  else
                        parse (String.sub s 1 (String.length(s)-1)) (word
^ (String.make 1 s.[0])) t
            in
                  parse s1 "" []


      let change_doc_list l =
            let rec change_doc_list_h l s =
                  match l with
                  | [] -> s
                  | h::t -> [convert_to_word_list h]@(change_doc_list_h t
s)
            in
                  change_doc_list_h l []

      let add_word w id pos inv_index =
            let d = { doc_id = id; freq = 1; positions = [pos]; tf = 0.0;}
in
```

```
            let w = { word = w; idf = 0.0; docs = [d]; } in
            inv_index @ [w]

      (*add a document in the inverted index corresponding to a word*)
      let add_doc w id pos inv_index =
            let match_word x = (x.word = w) in
            let (l1,l2) = List.partition match_word inv_index in
            if(List.length l1 == 0) then add_word w id pos inv_index
            else
                  let h = List.hd l1 in
                  let docs = h.docs in
                  let is_present x = (x.doc_id = id) in
                  if(List.exists is_present docs) then
                        let rec update_doc_info doc_list id pos
new_doc_list =
                              match doc_list with
                              | [] -> new_doc_list
                              | h::t -> if(h.doc_id == id) then
                                                let x = {doc_id = h.doc_id;
freq = h.freq+1; positions = h.positions@[pos]; tf = 0.0;} in
                                                update_doc_info t id pos
new_doc_list@[x]
                                            else
                                                update_doc_info t id pos
new_doc_list@[h]
                        in
                        let new_docs = update_doc_info docs id pos [] in
                        let new_word_entry = { word = w; idf = 0.0; docs =
new_docs; } in
                        l2@[new_word_entry]
                  else
                        let d = { doc_id = id; freq = 1; positions =
[pos]; tf = 0.0;} in
                        let new_docs = docs@[d] in
                        let new_word_entry = { word = w; idf = 0.0; docs =
new_docs; } in
                        l2@[new_word_entry]

      (* given a list of words corresponding to a doc id, update inverted
index *)
      let rec update_index words id inv_index =
            match words with
            | [] -> inv_index
            | (word,pos)::t ->
                  let new_index = add_doc word id pos inv_index in
                  update_index t id new_index

      let create_inv_index docs =
            let rec create_inv_index_h docs id index=
                  match docs with
                  | [] -> index
                  | h::t ->
                        let new_index = update_index h id index in
                        create_inv_index_h t (id+1) new_index
            in
                  create_inv_index_h docs 0 []

      let get_total_terms docs id =
            let d = List.nth docs id in
            List.length d
      let rec update_tfidf inv_index docs new_inv_index =
            match inv_index with
            | [] -> new_inv_index
            | h::t ->
                  let doc_list = h.docs in
                  let n_docs = List.length docs in
                  let rec update_tf l new_list =
                        match l with
                        | [] -> new_list
                        | h::t ->
                              let n_terms = get_total_terms docs h.doc_id
in
                              let x = {doc_id = h.doc_id;freq = h.freq;
positions = h.positions; tf = (float_of_int h.freq)/.( float_of_int
n_terms) ;} in
                              update_tf t new_list@[x]
                  in
                  let new_doc_list = update_tf doc_list [] in
                  let n_docs_occur = List.length doc_list in
                  let idf_score = log10((float_of_int
n_docs)/.(float_of_int n_docs_occur)) in
                  let w = {word = h.word; idf = idf_score; docs = List.rev
(new_doc_list);} in
                  update_tfidf t docs new_inv_index@[w]

      (*sort the inverted index alphabetically*)
      let comparator a b = compare (a.word) (b.word)

      let sort_index inv_index = List.sort comparator inv_index

      let rec print_int_list l =
            match l with
            | [] -> ()
            | h::[] -> print_int h
            | h::t -> print_int h; print_string ","; print_int_list t

      let print_doc_info d =
            print_string "( ";
            print_int d.doc_id;
            print_string ", ";
            print_int d.freq;
            print_string ", ";
            print_float d.tf;
            print_string ", [";
            print_int_list d.positions;
            print_string "])"

      let rec print_index l =
            match l with
            | [] -> ()
            | h::t ->
                   let docs = h.docs in
                   let rec print_docs d =
                         match d with
                         | [] -> ()
                         | h::[] -> print_doc_info h
                         | h::t -> print_doc_info h; print_string ", ";
print_docs t
                   in
```

```
                    print_string (h.word); print_string ": idf = ";
print_float h.idf; print_string ", docs: ["; print_docs docs;
print_string "]\n"; print_index t

      let rec make_string lt  = match lt with
                                          [] -> []
                                          | hd::tl ->  (string_of_int
hd)::make_string tl ;;

      let rec tuple_to_string tuple_list = match tuple_list with
                                          [] -> []
                                          | (s,l)::tl -> (String.concat " "
(s ::(make_string l))) :: (tuple_to_string tl);;

      let rec make_poslist_to_String poslist = match poslist with
                                                            [] -> ""
                                                            |hd :: [] -
>(string_of_int hd)
                                                            | hd::tl ->
(string_of_int hd)^";"^(make_poslist_to_String tl);;
      let rec make_doc_list_to_String doc_list = match doc_list with
                                                                  [] -> ""
                                                                  |hd ::[] -
>"("^(string_of_int hd.doc_id)^":"^(string_of_float
hd.tf)^":"^"["^(make_poslist_to_String hd.positions)^"])"
                                                                  | hd::tl ->
"("^(string_of_int hd.doc_id)^":"^(string_of_float
hd.tf)^":"^"["^(make_poslist_to_String
hd.positions)^"])"^","^(make_doc_list_to_String tl);;
      let rec convert_final_to_string final = match final with
                                                      [] -> []
                                                      | h::tl -> (h.word^"
"^(string_of_float h.idf)^" ["^(make_doc_list_to_String h.docs)^"]" )::
(convert_final_to_string tl);;


      (*let s = ["I am Anmol.  I Am the TA for col100."; "I am in my
final year"; "I love to swim"];;*)
      let final_list =read_filedoc in_file;;
      let l = change_doc_list final_list;;

      let inv_index = create_inv_index l;;
      let tfidf = update_tfidf inv_index l []
      let final = sort_index tfidf;;

      let final_string_l = convert_final_to_string final;;
      (*change "out.txt" to Sys.argv.(2)*)
      let output_channel = open_out out_file in
      List.map (Printf.fprintf output_channel "%s\n") final_string_l;
      close_out output_channel;

      (*print_index final;*)
```