

# BmkTrans: bookmarks pretty-printing

Pierre Boulet ([Pierre.Boulet@lifl.fr](mailto:Pierre.Boulet@lifl.fr))

Version 3.3

# Contents

<b>1 Documentation</b>	<b>2</b>
1.1 Usage: bmktrans -help	2
1.2 Installation	2
1.2.1 Requirements	2
1.2.2 Compilation	2
1.3 History	3
<b>2 Implementation</b>	<b>4</b>
2.1 Code structure	4
2.2 Module Analyze signature	4
2.3 Module Analyze implementation	5
2.3.1 Type definitions	5
2.3.2 Lexer	5
2.3.3 Grammar	9
2.3.4 htmldoc manipulation	11
2.4 Module Print signature	12
2.5 Module Print implementation	13
2.5.1 Basic printers	13
2.5.2 Hierarchy production	14
2.5.3 File description set generation	15
2.5.4 Circular list of colors	16
2.5.5 File description printer	17
2.6 Standalone program bmktrans	20

# Chapter 1

## Documentation

### 1.1 Usage: `bmktrans -help`

### 1.2 Installation

#### 1.2.1 Requirements

The following programs are required to compile BmkTrans

- Objective Caml version 3.08 or higher
- Camlp4 version corresponding to that of ocaml
- GNU Make

If you want to produce this documentation, you will need

- MIDoc, that you can find at the same place you've found BmkTrans
- $\text{\LaTeX}2\text{e}$  with packages `alltt` and `color`; packages `fontenc`, `inputenc` and `fullpage` are needed to use option `-main`
- the bera fonts for  $\text{\LaTeX}2\text{e}$
- $\text{\HEVEA}$  version 1.5 or above (<http://para.inria.fr/~maranget/hevea/>)

#### 1.2.2 Compilation

First check the `Makefile.config` file for customization. Then, type `make` and `make doc` to generate this documentation in the `doc` directory.

In case of strange errors, type `make depend` to regenerate the `.depend` dependency file. The one that comes with the source accomodates an installation of ocaml v3.04 or higher under `/usr`.

To install, just copy the produced `bmktrans` file where you want. The `doc` directory contains the postscript and html documentation.

### 1.3 History

v 3.3 (30 aug 2005)	compile with ocaml v3.08 (location treatment in camlp4) handle LAST_MODIFIED attribute in H1
v 3.2 (16 dec 2004)	add attributes SHORTCUTURL and LAST_MODIFIED html tag bug in titles and comments corrected
v 3.1 (19 mar 2003)	adapted to mozilla's file format META lines replicated to each output file bug correction in html rendering of titles and comments
v 3.0 (15 mar 2002)	new option -css and cleanup of the implementation
v 2.6 (23 jan 2002)	valid DOCTYPE, charset and HTML 4.01 transitional compliant
v 2.5 (15 dec 2000)	META and BASE tags are now correctly skipped in the header documentation generation by MIDoc
v 2.4 (12 dec 2000)	the document title is now displayed on each page
v 2.3 (11 dec 2000)	bug correction: create destination directory if it doesn't exist
v 2.2 (5 sep 2000)	deal with aliases by ignoring them ocaml and camlp4 v3.x required
v 2.1 (7 mar 2000)	new option -separator to set a custom separator between the URL and its description
v 2.0 (3 mar 2000)	take into account the description field highlight the new bookmarks and folders

# Chapter 2

## Implementation

### 2.1 Code structure

The code is divided in 3 modules:

- the analyze module parses the input bookmark file
- the print module defines all the pretty-printing functions
- the bmktrans handles the command-line arguments and calls the parser and pretty-printer.

### 2.2 Module Analyze signature

Parsing of a netscape (or mozilla or gnobog) bookmark file.

---

```
type header = {hd_contenu : string;
                hd_description : string;
                hd_add : float}
type href = {href_url : string;
              href_contenu : string;
              href_description : string;
              href_icon : string;
              href_add : float}
type liste = {l_header : header; l_refs : ligne list}
and ligne = Href of href | Liste of liste | Separator | Filtered_out
type htmldoc = {meta: string list; titre : string; bmk : ligne list; footer : string}

val analyze : string -> htmldoc
```

The *filter* function removes the folders whose titles match one of a given keyword set.

---

```
module Keywords :
  sig
    type t
    val empty : t
```

```

    val add : string -> t -> t
    val mem : string -> t -> bool
end

val filter : htmldoc -> Keywords.t -> htmldoc

```

The `new_top` function sets the top of the hierarchy to the first folder named as the given string.

---

```

val new_top : htmldoc -> string -> htmldoc

```

## 2.3 Module Analyze implementation

### 2.3.1 Type definitions

---

```

open Stdpp
open Token
open Grammar

type header = {hd_contenu : string;
               hd_description : string;
               hd_add : float}
type href = {href_url : string;
             href_contenu : string;
             href_description : string;
             href_icon : string;
             href_add : float}
type liste = {l_header : header; l_refs : ligne list}
and ligne = Href of href | Liste of liste | Separator | Filtered_out
type htmldoc = {meta: string list; titre : string; bmk : ligne list; footer : string}

```

### 2.3.2 Lexer

It is largely inspired from file `plexer.ml` of `camlp4` distribution

---

```

(* The string buffering machinery *)
let buff = ref (String.create 80)
let store len x =
  if len >= String.length !buff then
    buff := !buff ^ String.create (String.length !buff);
  !buff.[len] <- x;
  succ len

let mstore len s =
  let rec add_rec len i =
    if i == String.length s then len else add_rec (store len s.[i]) (succ i)

```

```

in
  add_rec len 0

let get_buff len = String.sub !buff 0 len

(* The lexer *)
let valch x = Char.code x - Char.code '0'

let rec ident len =
  parser
    [< ' (!' | '#'..'';' | '=' | '?'..'\'255' as c); s >] ->
      ident (store len c) s
    | [< ' '>' >] -> store len '>'
    | [< >] -> len

let next_token_fun find_id_kwd find_spe_kwd =
  let err bp ep msg = raise_with_loc (make_loc (bp, ep)) (Token.Error msg) in
  let keyword_or_error (bp, ep) c s =
    let spe = get_buff (ident (store 0 c) s) in
    begin try ("", (find_spe_kwd spe)) with
      Not_found -> err bp ep ("illegal token: " ^ spe)
    end
  in
  let rec next_token =

    parser bp
      [< ' (!' | '#'..'';' | '=' | '?'..'\'255' as c); s >] ->
        let id = get_buff (ident (store 0 c) s) in
        begin try ("", (find_id_kwd id)) with
          Not_found -> ("LIDENT", id)
        end
      | [< ' '>' >] ->
        begin try ("", (find_id_kwd ">")) with
          Not_found -> ("LIDENT", ">")
        end
      | [< ' ''' ; s >] -> ("STRING", (string bp 0 s))
  and string bp len =
    parser
      [< ' ''' >] -> get_buff len
      | [< 'c; s >] -> string bp (store len c) s
      | [< >] ep -> err bp ep "string not terminated"
  in
  let rec next_token_loc =
    parser bp
      [< ' '\n' | '\r' | '\t' | '\026' | '\012'; s >] ->
        next_token_loc s
      | [< ' '<; s >] -> maybe_comment bp s
      | [< tok = next_token >] ep -> (tok, (bp, ep))
      | [< _ = Stream.empty >] -> (("EOI", ""), (bp, succ bp))
  and maybe_comment bp =
    parser
      [< ' '!'; s >] -> comment bp s; next_token_loc s
      | [< s >] ep -> let tok = keyword_or_error (bp, ep) '<' s in
        (tok, (bp, ep))
  and comment bp =
    parser
      [< ' '>' >] -> ()
      | [< 'c; s >] -> comment bp s

```

```

| [< >] ep -> err bp ep "comment not terminated"
in
  fun cstrm ->
    try next_token_loc cstrm with
      Stream.Error str ->
        err (Stream.count cstrm) ((Stream.count cstrm)+1) str

let locerr () = invalid_arg "Lexer: location function"
let loct_create () = ref (Array.create 1024 None)
let loct_func loct i =
  match
    if i < 0 || i >= Array.length !loct then None
    else Array.unsafe_get !loct i
  with
    Some loc -> make_loc loc
  | _ -> locerr ()

let loct_add loct i loc =
  if i >= Array.length !loct then
    begin
      let new_tmax = Array.length !loct * 2 in
      let new_loct = Array.create new_tmax None in
      Array.blit !loct 0 new_loct 0 (Array.length !loct); loct := new_loct; ()
    end;
  !loct.(i) <- Some loc;
  ()

let func kwd_table =
  let find = Hashtbl.find kwd_table in
  let lex cstrm =
    let next_token_loc = next_token_fun find in
    let loct = loct_create () in
    let ts =
      Stream.from
        (fun i ->
          let (tok, loc) = next_token_loc cstrm in
          loct_add loct i loc; Some tok)
    in
    let locf = loct_func loct in ts, locf
  in
  lex

let check_keyword kwd = true

let using_token kwd_table (p_con, p_prm) =
  match p_con with
  "" ->
    begin try let _ = Hashtbl.find kwd_table p_prm in () with
      Not_found ->
        if check_keyword p_prm then Hashtbl.add kwd_table p_prm p_prm
        else
          raise
            (Token.Error
              ("the token \"\" ^ p_prm ^
              \"\" does not respect Plexer rules"))
    end
  | "LIDENT" | "UIDENT" | "INT" | "FLOAT" | "CHAR" | "STRING" | "QUOTATION" |
    "ANTIQUOT" | "LOCATE" | "EOI" ->

```

```

    ()
  | _ ->
    raise
      (Token.Error
        ("the constructor \"\" ^ p_con ^ \"\" is not recognized by Plexer"))

let removing_token kwd_table (p_con, p_prm) =
  if p_con = "" then Hashtbl.remove kwd_table p_prm

let text =
  function
    "", t -> "" ^ t ^ ""
  | "LIDENT", "" -> "lowercase identifier"
  | "LIDENT", t -> "" ^ t ^ ""
  | "UIDENT", "" -> "uppercase identifier"
  | "UIDENT", t -> "" ^ t ^ ""
  | "INT", "" -> "integer"
  | "INT", s -> "" ^ s ^ ""
  | "FLOAT", "" -> "float"
  | "STRING", "" -> "string"
  | "CHAR", "" -> "char"
  | "QUOTATION", "" -> "quotation"
  | "ANTIQUOT", k -> "antiquot \"\" ^ k ^ \"\""
  | "LOCATE", "" -> "locate"
  | "EOI", "" -> "end of input"
  | con, "" -> con
  | con, prm -> con ^ " \"\" ^ prm ^ \"\""

let eq_before_colon p e =
  let rec loop i =
    if i == String.length e then
      failwith "Internal error in Plexer: incorrect ANTIQUOT"
    else if i == String.length p then e.[i] == ':'
    else if p.[i] == e.[i] then loop (i + 1)
    else false
  in
  loop 0

let after_colon e =
  try
    let i = String.index e ':' in
      String.sub e (i + 1) (String.length e - i - 1)
  with
    Not_found -> ""

let tparse =
  function
    "ANTIQUOT", p_prm ->
      (fun (strm__ : _ Stream.t) ->
        match Stream.peek strm__ with
        Some ("ANTIQUOT", prm) when eq_before_colon p_prm prm ->
          Stream.junk strm__; after_colon prm
        | _ -> raise Stream.Failure)

```

```

| p_con, "" ->
  (fun (strm__ : _ Stream.t) ->
    match Stream.peek strm__ with
    | Some (con, prm) when con = p_con -> Stream.junk strm__; prm
    | _ -> raise Stream.Failure)
| p_con, p_prm ->
  fun (strm__ : _ Stream.t) ->
    match Stream.peek strm__ with
    | Some (con, prm) when con = p_con && prm = p_prm ->
      Stream.junk strm__; prm
    | _ -> raise Stream.Failure

```

```
let tparse _ = None
```

```

let lexer =
  let kwd_table = Hashtbl.create 301 in
  {func = func kwd_table; using = using_token kwd_table;
   removing = removing_token kwd_table; tparse = tparse; text = text}

```

### 2.3.3 Grammar

```

let gram = Grammar.create lexer
let skip = Grammar.Entry.create gram "skip"
let meta = Grammar.Entry.create gram "meta"
let title = Grammar.Entry.create gram "title"
let header = Grammar.Entry.create gram "header"
let att = Grammar.Entry.create gram "att"
let att_name = Grammar.Entry.create gram "att_name"
let liste = Grammar.Entry.create gram "liste"
let ligne = Grammar.Entry.create gram "ligne"
let fin_ligne = Grammar.Entry.create gram "fin_ligne"
let document = Grammar.Entry.create gram "document"
let termes = Grammar.Entry.create gram "termes"
let terme = Grammar.Entry.create gram "terme"
let alias = Grammar.Entry.create gram "alias"

```

#### EXTEND

```

document: [ [ LISTO skip; m = LISTO meta; t = title;
             "<DL>"; "<p>"; b = LISTO ligne; "</DL>"; "<p>"; EOI
             -> {meta=m; titre=t; bmkb=b; footer=""} ] ];
skip: [ [ "<BASE" | terme ] ];
meta: [ [ "<META"; m = termes; ">" -> "<META ^m^>\n" ] ];
title: [ [ "<TITLE>"; s = termes; "</TITLE>";
           "<H1>"; s' = termes; "</H1>"; OPT "<DD>" -> s |
           "<TITLE>"; s = termes; "</TITLE>";
           "<H1"; OPT "LAST_MODIFIED="; OPT STRING; ">"; s' = termes; "</H1>"; OPT "<DD>" -> s ] ];
termes: [ [ l = LISTO terme -> String.concat " " l ] ];
terme: [ [ s = LIDENT -> s
          | s = STRING -> s ] ];
liste: [ [ h = header; "<DL>"; "<p>"; l = LISTO ligne; "</DL>"; "<p>" ->
          {l_header = h; l_refs = l} ] ];
att_name: [ [ "FOLDED" -> ()
             | "NEWITEMHEADER" -> ()
             | "PERSONAL_TOOLBAR_FOLDER=" -> ()

```

```

| "NEW_BOOKMARK_FOLDER=" -> ()
| "ID=" -> ()
| "SHORTCUTURL=" -> ()
| "LAST_VISIT=" -> ()
| "LAST_CHARSET=" -> () ] ];
att: [ [ att_name; OPT STRING -> () ] ];
header: [ [ "<H3"; LIST0 att;
OPT "ADD_DATE="; add = OPT STRING;
LIST0 att;
OPT "LAST_MODIFIED="; lm = OPT STRING;
LIST0 att;
">"; s = termes; "</H3>";
OPT "<DD>"; d = termes ->
{hd_contenu = s;
hd_description = d;
hd_add = match add,lm with
| Some n, Some n' -> float_of_string (max n n')
| Some n, None -> float_of_string n
| None, Some n -> float_of_string n
| None, None -> 0.} ] ];
ligne: [ [ "<DT>"; l = fin_ligne -> l
| "<HR>" -> Separator ] ];
fin_ligne: [ [ l = liste -> Liste l
| "<A"; "HREF="; url = STRING;
OPT alias;
LIST0 att;
OPT "ADD_DATE="; add = OPT STRING;
LIST0 att;
OPT "LAST_MODIFIED="; lm = OPT STRING;
OPT "ICON="; icon = OPT STRING;
LIST0 att; ">";
s = termes; "</A>"; OPT "<DD>"; d = termes ->
Href {href_url = url;
href_contenu = s;
href_description = d;
href_icon = (match icon with
| None -> ""
| Some s -> s);
href_add = match add,lm with
| Some n, Some n' -> float_of_string (max n n')
| Some n, None -> float_of_string n
| None, Some n -> float_of_string n
| None, None -> 0.} ] ];
alias: [ [ "ALIASID="; STRING -> ()
| "ALIASOF="; STRING -> () ] ];

```

**END**

```

let analyze filename =
let ch = open_in filename in
try
let d = Grammar.Entry.parse document (Stream.of_channel ch)
in close_in ch; d
with
Stdpp.Exc_located (loc, e) ->
let (realfile, line, bc, ec) = line_of_loc filename loc in
Printf.printf "File %s, line %d, characters %d-%d\n"
filename line bc ec;
raise e

```

### 2.3.4 htmldoc manipulation

Html reconstruction in labels.

---

```
let rehtml ref =
  let rehtml' st =
    List.fold_left
      (fun s kw ->
        Str.global_replace
          (Str.regexp ("&lt;"^kw^"&gt;\\(.*\\)&lt;/"^kw^"&gt;"))
          ("<"^kw^">\\1</"^kw^">") s)
      st
    ["strong"; "em"; "code"; "b"; "i"; "tt"] in
  {href_url = ref.href_url;
   href_contenu = rehtml' ref.href_contenu;
   href_description = rehtml' ref.href_description;
   href_icon = ref.href_icon;
   href_add = ref.href_add}
```

Keywords out filtering.

---

```
module Keywords =
  Set.Make (struct
    type t = string
    let compare = compare
  end)

let remove_filtered_out l =
  List.fold_right
    (fun ligne clean_list -> match ligne with
      Filtered_out -> clean_list
      | _ -> ligne::clean_list)
    l
  []

let rec filtre_ligne kwds = function
  Liste {l_header = h; l_refs = r} ->
    if (Keywords.mem h.hd_contenu kwds) then Filtered_out
    else Liste {l_header = h;
                l_refs = remove_filtered_out
                        (List.map (filtre_ligne kwds) r)}
  | Href h -> Href (rehtml h)
  | l -> l

let filter doc kwds =
  {meta = doc.meta;
   titre = doc.titre;
   bmk = remove_filtered_out
         (List.map (filtre_ligne kwds) doc.bmk);
   footer = doc.footer}
```

Top folder setting.

---

```
let rec top_ligne top r l =
  match r with
  [] ->
    begin match l with
      Liste {l_header = h; l_refs = re} ->
        if h.hd_contenu = top then re
        else List.fold_left (top_ligne top) [] re
      | _ -> []
    end
  | _ -> r

let new_top doc top =
  {meta = doc.meta;
  titre = doc.titre;
  bmk = (List.fold_left (top_ligne top) [] doc.bmk);
  footer = doc.footer}
```

## 2.4 Module Print signature

Definition of the basic printers.

---

```
val print_href : Analyze.href -> unit
val print_header : Analyze.header -> unit
val print_liste : Analyze.liste -> unit
val print_ligne : Analyze.ligne -> unit
val print_html doc : Analyze.html doc -> unit
```

Object machinery to handle circular lists of colors.

---

```
class virtual duplicatable :
  object ('a)
    method virtual dup : 'a
  end;;

class color : string ->
  object ('a)
    method col : string
    method dup : 'a
  end

class ['a] circular_list :
  object ('b)
    constraint 'a = #duplicatable
    method add : 'a -> unit
    method get : 'a
    method next : 'a
```

```

    method dup : 'b
    method clear : unit
end

type colors = (color circular_list) circular_list

```

The main function, given a document and all the rendering options, it generates the hierarchy of html files.

---

```

val hierarchie :
  Analyze.html doc ->
  title:string option ->
  footer:string option -> dir:string -> depth:int ->
  css:string option -> bg:string -> text:string ->
  link:string -> vlink:string -> alink:string ->
  bar_width:int -> bar:colors -> list:colors ->
  visibility:bool -> new_age:float -> new_start:string ->
  new_end:string -> separator:string
  -> unit

```

## 2.5 Module Print implementation

### 2.5.1 Basic printers

---

```

open Analyze
open Format

let tab = 1

let print_href h =
  open_box tab;
  print_string ("<dt><a href=\"\"^h.href_url^\">");
  print_cut ();
  print_string h.href_contenu;
  print_cut ();
  print_string "</a></dt>";
  close_box ()

let print_header h =
  open_box tab;
  print_string ("");
  print_cut ();
  print_string h.hd_contenu;
  print_space ();
  print_string h.hd_description;
  print_cut ();
  print_string ("");
  close_box ()

let rec print_liste l =

```

```

open_box tab;
  print_string "<dt>";
  print_cut ();
  print_header l.l_header;
  print_cut ();
  print_string "</dt>";
  print_cut ();
  print_string "<dd><dl>";
  open_vbox 0;
    print_cut ();
    List.iter print_ligne l.l_refs;
  close_box ();
  print_string "</dl></dd>";
close_box ()
and print_ligne li =
open_box tab;
  begin
    match li with
      Href h -> print_href h
    | Liste l -> print_liste l
    | Separator -> print_string "<hr>"
    | Filtered_out -> print_string "<dt>Filtered out</dt>"
  end;
close_box ();
print_cut ()

let print_html doc d =
open_vbox 0;
  List.iter print_string d.meta;
  print_string ("<title>^d.titre^</title>");
  print_cut ();
  print_string ("<h1>^d.titre^</h1>");
  print_cut ();
  print_string "<dl>";
  print_cut ();
  List.iter print_ligne d.bmk;
  print_cut ();
  print_string "</dl>";
  print_cut ();
  print_string d.footer;
close_box ()

```

## 2.5.2 Hierarchy production

Headings generation.

---

```

type entête = {enom : string; enum : int}
type fichier = {tête : entête list; corps : ligne list}

let name_of_entête = function
  [] -> "index.html"
  | l ->
    let rec name = function
      [] -> ""
      | t::q -> (name q)^"_"^(string_of_int t.enum)

```

```

    in (name l)^".html"

let print_entête f e =
  let rec refer f = function
    [] -> ()
  | t::q -> begin
      (refer f q);
      pp_print_string f "<a href=\"";
      pp_print_string f (name_of_entête (t::q));
      pp_print_string f "\">";
      pp_print_string f t.enom;
      pp_print_string f "</a> >";
      pp_print_space f ()
    end
  in match e with
    [] -> pp_print_string f "<p>[Top]</p>"
  | _ -> pp_open_box f 2;
      pp_print_string f "<p><a href=\"index.html\">[Top]</a> >";
      pp_print_space f ();
      refer f (List.tl e);
      pp_print_space f ();
      pp_print_string f (List.hd e).enom;
      pp_print_string f "</p>";
      pp_close_box f ()

```

### 2.5.3 File description set generation

The *parcours* creates the set of file descriptions (type *fichier*) to generate.

---

```

module Files =
  Set.Make (struct
    type t = fichier
    let compare = compare
  end)

class compteur =
  object
    val mutable c = 0
    method get = c <- c + 1; c
  end

let rec tronque p lignes =
  if p = 1 then
    List.map
      (function
        Liste l -> Liste {l_header = l.l_header;
                          l_refs = []})
      | l -> l)
    lignes
  else List.map
      (function
        Liste l -> Liste {l_header = l.l_header;
                          l_refs = (tronque (p-1) l.l_refs)})
      | l -> l)
    lignes

```

```

let rec parcours set p entête lignes =
  let c = new compteur in
  List.fold_left
    (fun set ligne -> match ligne with
      Liste l ->
        parcours set p
          ({enom = l.l_header.hd_contenu; enum = c#get}::entête)
          l.l_refs
      | _ -> set)
    (Files.add {tête = entête; corps = tronque p lignes} set)
  lignes

```

## 2.5.4 Circular list of colors

---

```

class virtual duplicatable =
  object (_ : 'a)
    method virtual dup : 'a
  end

class color (c_init : string) =
  object (self)
    inherit duplicatable
    val c = c_init
    method col = c
    method dup = 0o.copy self
  end

```

We need to use a *dup* method to be able to physically replicate the list. If we just use *0o.copy*, the internal queue is shared between the two copies. This is needed to have a consistent look for all the files.

---

```

class ['a] circular_list =
  object (self)
    inherit duplicatable
    constraint 'a = #duplicatable
    val mutable cols : ('a Queue.t) option = None
    method add c = match cols with
      Some q -> Queue.add c q
    | None ->
      cols <- Some (Queue.create ());
      self#add c
    method next = match cols with
      None -> raise Queue.Empty
    | Some q ->
      let c = Queue.take q in
      self#add c;
      c
    method get = match cols with
      None -> raise Queue.Empty
    | Some q -> Queue.peek q

```

```

method dup =
  let d = 0o.copy self in
    d#clear;
    match cols with
      None -> d
    | Some q ->
      for i = 1 to Queue.length q do
        d#add self#next#dup
      done;
      d
method clear = cols <- None
end

type colors = (color circular_list) circular_list

```

## 2.5.5 File description printer

---

```

let rec print_lignes
  f n entête ep_barre
  col_barre col_liste invisible
  add_limit new_marker_start new_marker_end separator
  ligne =
  let inv_col = col_liste#get#get#col in
    pp_open_vbox f 0;
    pp_print_string f
      ("|<td bgcolor=\"\"^col_liste#get#next#col^\">");
    pp_print_break f 0 2;
    begin match ligne with
      Href h ->
        pp_open_box f 2;
        if h.href_add >= add_limit then
          pp_print_string f new_marker_start;
          pp_print_string f "<a"; pp_print_space f ();
          pp_print_string f ("href=\"\"^h.href_url^\">");
          pp_print_cut f ();
          (*
          pp_print_string f ("<img href=\"\"^h.href_icon^\">");
          pp_print_cut f ();
          *)
          pp_print_string f h.href_contenu;
          pp_print_cut f ();
          pp_print_string f "</a>";
          if h.href_add >= add_limit then
            pp_print_string f new_marker_end;
            if h.href_description <> "" then begin
              pp_print_string f separator;
              pp_print_cut f ();
              pp_print_string f h.href_description
            end;
            pp_close_box f ();
        | Liste {l_header = h; l_refs = l} ->
          let e = {enom = h.hd_contenu; enum = n#get}::entête
          in pp_open_vbox f 2;
          if h.hd_add >= add_limit then
            pp_print_string f new_marker_start;
            pp_print_string f
              ("<font size=\"+1\"><strong><a href=\"\"^(name_of_entête e)


```

```

        ^"\>"^h.hd_contenu^"</a></strong></font>");
    if h.hd_add >= add_limit then
        pp_print_string f new_marker_end;
    if h.hd_description <> "" then begin
        pp_print_string f separator;
        pp_print_cut f ();
        pp_print_string f h.hd_description
    end;
    pp_print_cut f ();
    print_table f e ep_barre
        col_barre col_liste invisible inv_col
        add_limit new_marker_start new_marker_end separator
        l;
    pp_close_box f ()
| Separator ->
    pp_print_string f "<hr>";
    pp_print_cut f ()
| Filtered_out -> failwith "filter"
end;
pp_print_break f 0 2;
pp_print_string f "</td></tr>";
pp_close_box f ()
and print_table f entête ep_barre
col_barre col_liste invisible inv_col
add_limit new_marker_start new_marker_end separator =
function
    [] -> ()
| l ->
    pp_open_vbox f 0;
    pp_print_string f ("<table bgcolor=\"\"^
        (col_liste#get#get#col)^\">");
    pp_print_break f 0 2;
    pp_open_vbox f 0;
    pp_print_string f "<tr>";
    pp_print_break f 0 2;
    pp_open_vbox f 0;
    pp_print_string f "<td width=\"\"";
    pp_print_int f ep_barre;
    pp_print_string f
        ("\" bgcolor=\"\"^
        (if invisible then inv_col
         else col_barre#get#next#col)^
        "\><img src=\"1x1.gif\" alt=\"\" height=1 width=1</td>");
    pp_print_cut f ();
    pp_print_string f "<td>";
    pp_print_break f 0 2;
    pp_print_string f "<table>";
    let ccl = col_liste#dup
    and ccb = col_barre#dup in
    ccb#next;
    ccl#next;
    let n = new compteur in
        List.iter (fun li -> pp_print_break f 0 2;
            print_lignes
                f n entête ep_barre
                ccb ccl invisible
                add_limit new_marker_start new_marker_end
                separator li) l;

```

```

        pp_print_string f "</table>";
    pp_print_break f 0 (-2);
    pp_print_string f "</td>";
    pp_close_box f ();
    pp_print_break f 0 (-2);
    pp_print_string f "</tr>";
    pp_close_box f ();
    pp_print_break f 0 (-2);
    pp_print_string f "</table>";
    pp_close_box f ()

```

**let** print\_fichier

```

répertoire titre meta footer css
col_fond col_texte col_link col_vlink col_alink
ep_barre col_barre col_liste invisible
add_limit new_marker_start new_marker_end separator
fichier =

```

```

let oc = open_out (Filename.concat
                    répertoire
                    (name_of_entête fichier.tête))
in let f = make_formatter (output oc) (fun () -> flush oc)

```

```

in pp_open_vbox f 2;
    pp_print_string f "<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">"; pp_print_cut f ();
    pp_print_string f "<html>"; pp_print_cut f ();
    pp_open_vbox f 2;
    pp_print_string f "<head>"; pp_print_cut f ();
    List.iter (fun m -> pp_print_string f m; pp_print_cut f ()) meta;
    begin match css with
    | Some url ->
        pp_print_string f ("<link href=\"\"^url^
                            \"\" rel=\"stylesheet\" type=\"text/css\">");
        pp_print_cut f ()
    | None -> ()
    end;
    begin match fichier.tête with
    [] -> pp_print_string f ("<title>^titre^</title>")
    | l -> pp_print_string f ("<title>^titre^: "
                              ^((List.hd l).enom)^"</title>")
    end;
    pp_close_box f (); pp_print_cut f ();
    pp_print_string f "</head>"; pp_print_cut f ();
    pp_open_vbox f 2;
    begin match css with
    | Some _ -> pp_print_string f "<body>"
    | None ->
        pp_print_string f
            ("<body text=\"\"^col_texte
             ^\" link=\"\"^col_link
             ^\" vlink=\"\"^col_vlink
             ^\" alink=\"\"^col_alink
             ^\" bgcolor=\"\"^col_fond^\">")
    end;
    pp_print_cut f ();
    begin match fichier.tête with
    [] -> pp_print_string f ("<h2>^titre^</h2>")
    | l -> pp_print_string f ("<h2>^titre^: "
                              ^((List.hd l).enom)^"</h2>")
    end;

```

```

pp_print_cut f ();
print_entête f fichier.tête; pp_print_cut f ();
pp_open_vbox f 2;
pp_print_string f "<p>"; pp_print_cut f ();
print_table f fichier.tête ep_barre col_barre
  col_liste invisible col_fond
  add_limit new_marker_start new_marker_end separator
  fichier.corps;
print_entête f fichier.tête; pp_print_cut f ();
pp_print_string f footer;
pp_close_box f (); pp_print_cut f ();
pp_print_string f "</body>";
pp_close_box f (); pp_print_cut f ();
pp_print_string f "</html>";
pp_print_flush f ();
close_out oc

```

```

let hierarchie doc ~title ~footer ~dir ~depth
~css ~bg:col_fond ~text:col_texte ~link:col_link
~vlink:col_vlink ~alink:col_alink
~bar_width:ep_barre ~bar:col_barre ~list:col_liste
~visibility:invisible ~new_age:max_new_age
~new_start:new_marker_start ~new_end:new_marker_end ~separator
=
let titre = (match title with
  None -> doc.titre
  | Some t -> t)
and footer = (match footer with
  None -> doc.footer
  | Some f -> f)
and add_limit = (Unix.time ()) -. max_new_age
in
Files.iter
  (print_fichier dir titre doc.meta footer css
  col_fond col_texte col_link col_vlink col_alink
  ep_barre col_barre#dup col_liste#dup invisible
  add_limit new_marker_start new_marker_end separator)
  (parcours Files.empty depth [] doc.bmk)

```

## 2.6 Standalone program bmktrans

This program handles the command line options and calls the html file generator.

---

```

open Analyze
open Print

```

```

(*****)
(* configurable default options *)
(*****)
let directory = ref "bookmarks"
let title = ref None
let footer =
  ref (Some ("<address>Made with "^
    "<a href=\"http://www.lifl.fr/~boulet/softs.html\">"^

```

```

        "bmktrans</a>.</address>"))
let depth = ref 3
let background_color = ref "#FFFFFF"
let text_color = ref "#000000"
let link_color = ref "#660000"
let vlink_color = ref "#000066"
let alink_color = ref "#660066"
let bar_width = ref 40
let bar_colors = ref (new circular_list)
let bcl = new circular_list
let _ = bcl#add (new color "#009966")
let _ = !bar_colors#add bcl
let invisible_bar = ref false
let list_colors = ref (new circular_list)
let lcl = new circular_list
let _ = lcl#add (new color "#99FF99")
let _ = lcl#add (new color "#CCFFCC")
let _ = !list_colors#add lcl
let keywords = ref Keywords.empty
let top = ref None
(* maximal age in seconds to be marked as new *)
let max_new_age = ref (3600. *. 24. *. 15.)
let new_marker_start = ref ""
let new_marker_end = ref " <b><i>NEW</i></b>"
let separator = ref " "
let css = ref None

(*****)
(* DO NOT EDIT BELOW THIS POINT *)
(*****)
let new_cb = ref true
let new_cl = ref true

let set_options () =
  let inputfile = ref ""
  in Arg.parse
    [("-directory", Arg.String (fun d -> directory := d),
      "<dir>: set output directory to <dir>\n"^
      "\t<dir> must exist before the execution of bmktrans\n"^
      "\tdefault value: bookmarks");
     ("-title", Arg.String (fun t -> title := Some t),
      "<string>: set running document title to <string>\n"^
      "\tdefault value taken from input file");
     ("-footer", Arg.String (fun f -> footer := Some f),
      "<string>: set running document footer to <string>\n"^
      "\tdefault value: "^<address>Made with "^
      "<a href=\"http://www.lifl.fr/~boulet/softs.html\">^
      "bmktrans</a>.</address>");
     ("-depth", Arg.Int (fun d -> depth := d),
      "<int>: set the tree pruning depth to <int>\n"^
      "\tdefault value: 3");
     ("-css", Arg.String (fun f -> css := Some f),
      "<url>: use the given url as style sheet\n"^
      "\tinhibits the bgcolor, text, link, vlink, alink options\n"^
      "\tdefault value: None");
     ("-bgcolor", Arg.String (fun c -> background_color := c),
      "<color>: set the background color to <color>\n"^
      "\tdefault value: #FFFFFF)];

```

```

("-text", Arg.String (fun c -> text_color := c),
  "<color>: set the text color to <color>\n"^
  "\tdefault value: #000000");
("-link", Arg.String (fun c -> link_color := c),
  "<color>: set the non visited link color to <color>\n"^
  "\tdefault value: #660000");
("-vlink", Arg.String (fun c -> vlink_color := c),
  "<color>: set the visited link color to <color>\n"^
  "\tdefault value: #000066");
("-alink", Arg.String (fun c -> alink_color := c),
  "<color>: set the active link color to <color>\n"^
  "\tdefault value: #660066");
("-barwidth", Arg.Int (fun w -> bar_width := w),
  "<int>: set the bar width to <int>\n"^
  "\tdefault value: 40");
("-barcols",
  Arg.String
    (fun cs ->
      if cs = "invisible" then invisible_bar := true
      else begin
        if !new_cb then begin
          new_cb := false;
          bar_colors := new circular_list
        end;
        let cb = new circular_list in
          for i = 0 to ((String.length cs)/7)-1 do
            cb#add (new color (String.sub cs (i*7) 7))
          done;
          !bar_colors#add cb
        end),
      "<colors>: set the bar colors to <colors>\n"^
      "\ta value of invisible is possible\n"^
      "\tdefault value: #009966\n"^
      "\ttry using it several times...");
("-listcols",
  Arg.String (fun cs ->
    if !new_cl then begin
      new_cl := false;
      list_colors := new circular_list
    end;
    let cl = new circular_list in
      for i = 0 to ((String.length cs)/7)-1 do
        cl#add (new color (String.sub cs (i*7) 7))
      done;
      !list_colors#add cl),
    "<colors>: set the lists colors to <colors>\n"^
    "\tdefault value: #99FF99#CCFFCC\n"^
    "\ttry using it several times...");
("-top",
  Arg.String (fun s -> top := Some s),
  "<string>: set the root of the produced hierarchy");
("-filter",
  Arg.String (fun s -> keywords := Keywords.add s !keywords),
  "<string>: remove from the output any folder with that name\n"^
  "\tcan appear any number of times");
("-maxnewage",
  Arg.Float (fun f -> max_new_age := f),
  "<float>: set the maximal age (in seconds) to be marked as new\n"^

```

```

        "\tdefault value: 1296000 (15 days)");
    ("-newmarkerstart",
    Arg.String (fun s -> new_marker_start := s),
    "<string>: set the part of the new marker that is inserted\n"^
    "\tbefore the bookmark or the folder name\n"^
    "\tdefault value: \"\");
    ("-newmarkerend",
    Arg.String (fun s -> new_marker_end := s),
    "<string>: set the part of the new marker that is inserted\n"^
    "\tafter the bookmark or the folder name\n"^
    "\tdefault value: \" <b><i>NEW</i></b>\");
    ("-separator",
    Arg.String (fun s -> separator := s),
    "<string>: set the separator between the URL and its description\n"^
    "\tdefault value: \" \");]
    (fun f -> inputfile := f)
    ("usage: bmktrans [options] <file>\n"^
    "\t<file> has to obey the syntax of netscape's bookmarks file\n"^
    "\t[options] are described below with their default value\n"^
    "\n<color> syntax: <color> is in the hexadecimal form #RRVVB\n"^
    "\t<color> is a sequence of <color> (see example below)\n"^
    "options:");
!inputfile

```

Main function. It creates the output directory if necessary, then puts a 1x1 transparent gif in it and calls the parser followed by the generator.

---

```

let main () =
  let filename = set_options () in
  if not(Sys.file_exists !directory) then
    Unix.mkdir !directory 0o755;
  let pixel = Filename.concat !directory "1x1.gif" in
  if not(Sys.file_exists pixel)
  then begin
    let chpix = open_out pixel in
    output_string chpix
      "GIF89a\001\000\001\000i\000\000\000\000\000ÿÿÿÿÿÿÿÿ";
    output_string chpix
      "!p\014Made with GIMP\000!ù\004\001\n";
    output_string chpix
      "\000\000\000,\000\000\000\000\001";
    output_string chpix
      "\000\001\000\000\002\002D\001\000";
    close_out chpix
  end;
  let doc = filter (analyze filename) !keywords in
  let doc' = match !top with
  None -> doc
  | Some s -> new_top doc s in
  Print.hierarchie doc'
  ~title:!title ~footer:!footer
  ~dir:!directory ~depth:!depth
  ~css:!css ~bg:!background_color ~text:!text_color
  ~link:!link_color ~vlink:!vlink_color ~alink:!alink_color
  ~bar_width:!bar_width ~bar:!bar_colors ~list:!list_colors

```

```
~visibility:!invisible_bar ~new_age:!max_new_age  
~new_start:!new_marker_start ~new_end:!new_marker_end  
~separator:!separator
```

```
let _ = Printexc.catch main ()
```