Commons$_{pad}$ OCaml Library

Programmer's manual

Yoann Padioleau
yoann.padioleau@gmail.com

December 29, 2009

Copyright © 2009 Yoann Padioleau.
Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.3.

Short Contents

Contents

Chapter 1

Introduction

[1]

[2]

[3]

1.1 Features

[4]

[5]

This directory builds a common.cma library and also optionally multiple commons_xxx.cma small libraries. The reason not to just build a single one is that some functionnalities require external libraries (like Berkeley DB, MPI, etc) or special version of OCaml (like for the backtrace support) and I don't want to penalize the user by forcing him to install all those libs before being able to use some of my common helper functions. So, common.ml and other files offer convenient helpers that do not require to install anything. In some case I have directly included the code of those external libs when there are simple such as for ANSITerminal in ocamlextra/, and for dumper.ml I have even be further by inlining its code in common.ml so one can just do a open Common and have everything. Then if the user wants to, he can also leverage the other commons_xxx libraries by explicitely building them after he has installed the necessary external files.

[1]TODO: commons is OCaml library, complete standard library

[2]TODO: getting started, commons.cma

[3]TODO: why in single file ?

[4]TODO: the most original features: - quicheck - profile - regression testing - maybe use latex visual trick to say when good function

[5]TODO: make features stuff configure, Makefile.config cf readme.txt ? features.ml.cpp, MPI, Regexp, Backtrace yes sometimes cpp is useful cf also common_extra

```
For many configurable things we can use some flags in ml files,
and have some -xxx command line argument to set them or not,
but for other things flags are not enough as they will not remove
the header and linker dependencies in Makefiles. A solution is
to use cpp and pre-process many files that have such configuration
issue. Another solution is to centralize all the cpp issue in one
file, features.ml.cpp, that acts as a generic wrapper for other
librairies and depending on the configuration actually call
the external library or provide a fake empty services indicating
that the service is not present.
So you should have a ../configure that call cpp on features.ml.cpp
to set those linking-related configuration settings.
```

6

```
(* I put those functions here and not in common.ml to try to avoid
 * as much as possible dependencies in common.ml so I can more easily
 * make ocaml script that just do a load common.ml without the need
 * to load many other files (like dumper.ml, or ANSITerminal.ml and
 * other recursive dependencies).
 *
 * Note that you can still use the functions below from an open Common.
 * You don't need to do a 'open Common_extra'; loading the commons.cma is
 * enough to make the connexions.
 *)

set_link
```

1.2 Copyright

Commons$_{pad}$ is governed by the following copyright:

This manual is copyright 2009 Yoann Padioleau, and distributed under the terms of the GNU Free Documentation License version 1.3.

Function	**Chapter**	**Modules**
Main services	??	`parse_c.mli`

Table 1.1: Chapters and modules

1.3 Source organization

1.4 API organization

1.5 Acknowledgements

Maybe a few code was borrowed from Pixel (Pascal Rigaux) and Julia Lawall may have written a few helpers.

Thanks to:

- Richard Jones for his dumper module,
- Brian Hurt and Nicolas Cannasse for their dynArray module,
- Troestler Christophe for his ANSITerminal module,
- Sebastien ferre for his suffix tree module
- Jane street for their backtrace module

[6]TODO: `why in single file ?`

Part I

Common

Chapter 2

Overview

8 ⟨*common.mli* 8⟩≡

```
(*###########################################################################*)
(* Globals *)
(*###########################################################################*)
⟨common.mli globals 9a⟩

(*###########################################################################*)
(* Basic features *)
(*###########################################################################*)
⟨common.mli basic features 14⟩

(*###########################################################################*)
(* Basic types *)
(*###########################################################################*)
⟨common.mli for basic types 31a⟩

(*###########################################################################*)
(* Collection-like types *)
(*###########################################################################*)
⟨common.mli for collection types 46⟩

(*###########################################################################*)
(* Misc functions *)
(*###########################################################################*)
⟨common.mli misc 61a⟩

(*###########################################################################*)
(* Postlude *)
(*###########################################################################*)
⟨common.mli postlude 10c⟩
```

```
(* Some conventions:
 *
 * When I have some _xxx variables before some functions, it's
 * because I want to show that those functions internally use a global
 * variable. That does not mean I want people to modify this global.
 * In fact they are kind of private, but I still want to show them.
 * Maybe one day OCaml will have an effect type system so I don't need this.
 *
 * The variables that are called _init_xxx show the internal init
 * side effect of the module (like static var trick used in C/C++)
 *
 * Why not split the functionnalities of this file in different files ?
 * Because when I write ocaml script I want simply to load one
 * file, common.ml, and that's it. Cf common_extra.ml for more on this.
 *)
```

9a ⟨*common.mli globals* 9a⟩≡

```
(*****************************************************************************)
(* Flags *)
(*****************************************************************************)
⟨common.mli globals flags 9b⟩

(*****************************************************************************)
(* Flags and actions *)
(*****************************************************************************)
(* cf poslude *)

(*****************************************************************************)
(* Misc/test *)
(*****************************************************************************)
⟨common.mli misc/test 12⟩
```

9b ⟨*common.mli globals flags* 9b⟩≡

```
(* see the corresponding section for the use of those flags. See also
 * the "Flags and actions" section at the end of this file.
 *)

(* if set then will not do certain finalize so faster to go back in replay *)
val debugger : bool ref

type prof = PALL | PNONE | PSOME of string list
val profile : prof ref
val show_trace_profile : bool ref
val verbose_level : int ref

(* forbid pr2_once to do the once "optimisation" *)
val disable_pr2_once : bool ref



(* works with new_temp_file *)
val save_tmp_files : bool ref
```

10a ⟨*common.mli globals* 9a⟩+≡

```
(*****************************************************************************)
(* Module side effect *)
(*****************************************************************************)
(*
 * I define a few unit tests via some let _ = example (... = ...).
 * I also initialize the random seed, cf _init_random .
 * I also set Gc.stack_size, cf _init_gc_stack .
*)
```

10b ⟨*common.mli globals* 9a⟩+≡

```
(*****************************************************************************)
(* Semi globals *)
(*****************************************************************************)
(* cf the _xxx variables in this file *)
```

10c ⟨*common.mli postlude* 10c⟩≡

```
val cmdline_flags_devel : unit -> cmdline_options
val cmdline_flags_verbose : unit -> cmdline_options
val cmdline_flags_other : unit -> cmdline_options
```

10d ⟨*common.ml cmdline* 10d⟩≡

```

(* I put it inside a func as it can help to give a chance to
 * change the globals before getting the options as some
 * options sometimes may want to show the default value.
 *)
let cmdline_flags_devel () =
  [
    "-debugger",         Arg.Set debugger ,
    "   option to set if launched inside ocamldebug";
    "-profile",          Arg.Unit (fun () -> profile := PALL),
    "   gather timing information about important functions";
  ]
let cmdline_flags_verbose () =
  [
    "-verbose_level",  Arg.Set_int verbose_level,
    " <int> guess what";
    "-disable_pr2_once",     Arg.Set disable_pr2_once,
    "   to print more messages";
    "-show_trace_profile",          Arg.Set show_trace_profile,
    "   show trace";
  ]

let cmdline_flags_other () =
  [
    "-nocheck_stack",      Arg.Clear check_stack,
    " ";
    "-batch_mode", Arg.Set _batch_mode,
    " no interactivity"
  ]

(* potentially other common options but not yet integrated:

  "-timeout",        Arg.Set_int timeout,
  "  <sec> interrupt LFS or buggy external plugins";

  (* can't be factorized because of the $ cvs stuff, we want the date
   * of the main.ml file, not common.ml
   *)
  "-version",   Arg.Unit (fun () ->
    pr2 "version: _dollar_Date: 2008/06/14 00:54:22 _dollar_";
    raise (Common.UnixExit 0)
    ),
  "   guess what";

  "-shorthelp", Arg.Unit (fun () ->
    !short_usage_func();
    raise (Common.UnixExit 0)
  ),
  "    see short list of options";
  "-longhelp", Arg.Unit (fun () ->
    !long_usage_func();
    raise (Common.UnixExit 0)
    ),
  "-help", Arg.Unit (fun () ->
    !long_usage_func();
    raise (Common.UnixExit 0)
  ),
  " ";
  "--help", Arg.Unit (fun () ->
    !long_usage_func();
    raise (Common.UnixExit 0)
  ),
  " ";

*)

let cmdline_actions () =
  [
    "-test_check_stack", "  <limit>",
    mk_action_1_arg test_check_stack_size;
  ]
```

12 ⟨*common.mli misc/test* 12⟩≡

```
  val generic_print : 'a -> string -> string

  class ['a] olist :
    'a list ->
    object
      val xs : 'a list
      method fold : ('b -> 'a -> 'b) -> 'b -> 'b
      method view : 'a list
    end

  val typing_sux_test : unit -> unit

(*****************************************************************************)
(* Notes *)
(*****************************************************************************)



(* ------------------------------------------------------------------------ *)
(* Maybe could split common.ml and use include tricks as in ofullcommon.ml or
 * Jane Street core lib. But then harder to bundle simple scripts like my
 * make_full_linux_kernel.ml because would then need to pass all the files
 * either to ocamlc or either to some #load. Also as the code of many
 * functions depends on other functions from this common, it would
 * be tedious to add those dependencies. Here simpler (have just the
 * pb of the Prelude, but it's a small problem).
 *
 * pixel means code from Pascal Rigaux
 * julia means code from Julia Lawall
 *)
(* ------------------------------------------------------------------------ *)

(*****************************************************************************)
```

```
(* We use *)
(*****************************************************************************)
(*
 * modules:
 *   - Pervasives, of course
 *   - List
 *   - Str
 *   - Hashtbl
 *   - Format
 *   - Buffer
 *   - Unix and Sys
 *   - Arg
 *
 * functions:
 *   - =, <=, max min, abs, ...
 *   - List.rev, List.mem, List.partition,
 *   - List.fold*, List.concat, ...
 *   - Str.global_replace
 *   - Filename.is_relative
 *   - String.uppercase, String.lowercase
 *
 *
 * The Format library allows to hide passing an indent_level variable.
 * You use as usual the print_string function except that there is
 * this automatic indent_level variable handled for you (and maybe
 * more services). src: julia in coccinelle unparse_cocci.
 *
 * Extra packages
 *  - ocamlbdb
 *  - ocamlgtk, and gtksourceview
 *  - ocamlgl
 *  - ocamlpython
 *  - ocamlagrep
 *  - ocamlfuse
 *  - ocamlmpi
 *  - ocamlcalendar
 *
 *  - pcre
 *  - sdl
 *
 * Many functions in this file were inspired by Haskell or Lisp librairies.
 *)
```

Chapter 3

Basic

3.1 Pervasive types and operators

14 ⟨*common.mli basic features* 14⟩≡

```
(*****************************************************************************)
(* Pervasive types and operators *)
(*****************************************************************************)

type filename = string
type dirname = string

(* Trick in case you dont want to do an 'open Common' while still wanting
 * more pervasive types than the one in Pervasives. Just do the selective
 * open Common.BasicType.
 *)
module BasicType : sig
  type filename = string
end

(* Same spirit. Trick found in Jane Street core lib, but originated somewhere
 * else I think: the ability to open nested modules. *)
module Infix : sig
  val ( +> ) : 'a -> ('a -> 'b) -> 'b
  val ( =~ ) : string -> string -> bool
  val ( ==~ ) : string -> Str.regexp -> bool
end


(*
 * Another related trick, found via Jon Harrop to have an extended standard
 * lib is to do something like
 *
 * module List = struct
 *  include List
 *  val map2 : ...
 * end
 *
 * And then can put this "module extension" somewhere to open it.
 *)



(* This module defines the Timeout and UnixExit exceptions.
 * You  have to make sure that those exn are not intercepted. So
 * avoid exn handler such as try (...) with _ -> cos Timeout will not bubble up
 * enough. In such case, add a case before such as
 * with Timeout -> raise Timeout | _ -> ...
 * The same is true for UnixExit (see below).
 *)
```

3.2 Debugging, logging

15 ⟨*common.mli basic features* 14⟩+≡

```
(*****************************************************************************)
(* Debugging/logging *)
(*****************************************************************************)

val _tab_level_print: int ref
val indent_do : (unit -> 'a) -> 'a
val reset_pr_indent : unit -> unit

(* The following functions first indent _tab_level_print spaces.
 * They also add the _prefix_pr, for instance used in MPI to show which
 * worker is talking.
 * update: for pr2, it can also print into a log file.
 *
 * The use of 2 in pr2 is because 2 is under UNIX the second descriptor
 * which corresponds to stderr.
 *)
val _prefix_pr : string ref

val pr : string -> unit
val pr_no_nl : string -> unit
val pr_xxxxxxxxxxxxxxxxx : unit -> unit

(* pr2 print on stderr, but can also in addition print into a file *)
```

```
val _chan_pr2: out_channel option ref
val pr2 : string -> unit
val pr2_no_nl : string -> unit
val pr2_xxxxxxxxxxxxxxxxx : unit -> unit

(* use Dumper.dump *)
val pr2_gen: 'a -> unit
val dump: 'a -> string

(* see flag: val disable_pr2_once : bool ref *)
val _already_printed : (string, bool) Hashtbl.t
val pr2_once : string -> unit

val mk_pr2_wrappers: bool ref -> (string -> unit) * (string -> unit)


val redirect_stdout_opt : filename option -> (unit -> 'a) -> 'a
val redirect_stdout_stderr : filename -> (unit -> unit) -> unit
val redirect_stdin : filename -> (unit -> unit) -> unit
val redirect_stdin_opt : filename option -> (unit -> unit) -> unit

val with_pr2_to_string: (unit -> unit) -> string list

val fprintf : out_channel -> ('a, out_channel, unit) format -> 'a
val printf : ('a, out_channel, unit) format -> 'a
val eprintf : ('a, out_channel, unit) format -> 'a
val sprintf : ('a, unit, string) format -> 'a

(* alias *)
val spf : ('a, unit, string) format -> 'a

(* default = stderr *)
val _chan : out_channel ref
(* generate & use a /tmp/debugml-xxx file *)
val start_log_file : unit -> unit

(* see flag: val verbose_level : int ref *)
val log : string -> unit
val log2 : string -> unit
val log3 : string -> unit
val log4 : string -> unit

val if_log  : (unit -> unit) -> unit
val if_log2 : (unit -> unit) -> unit
val if_log3 : (unit -> unit) -> unit
val if_log4 : (unit -> unit) -> unit
val pause : unit -> unit

(* was used by fix_caml *)
val _trace_var : int ref
val add_var : unit -> unit
val dec_var : unit -> unit
val get_var : unit -> int

val print_n : int -> string -> unit
val printerr_n : int -> string -> unit

val _debug : bool ref
val debugon : unit -> unit
val debugoff : unit -> unit
val debug : (unit -> unit) -> unit

(* see flag: val debugger : bool ref *)
```

3.3 Profiling

17 ⟨*common.mli basic features* 14⟩+≡

```
(*****************************************************************************)
(* Profiling (cpu/mem) *)
(*****************************************************************************)

val get_mem : unit -> unit
val memory_stat : unit -> string

val timenow : unit -> string

val _count1 : int ref
val _count2 : int ref
val _count3 : int ref
val _count4 : int ref
val _count5 : int ref


val count1 : unit -> unit
val count2 : unit -> unit
val count3 : unit -> unit
val count4 : unit -> unit
val count5 : unit -> unit
val profile_diagnostic_basic : unit -> string
val time_func : (unit -> 'a) -> 'a



(* see flag: type prof = PALL | PNONE | PSOME of string list *)
(* see flag: val profile : prof ref *)

val _profile_table : (string, (float ref * int ref)) Hashtbl.t ref
val profile_code : string -> (unit -> 'a) -> 'a
val profile_diagnostic : unit -> string

val profile_code_exclusif : string -> (unit -> 'a) -> 'a
val profile_code_inside_exclusif_ok : string -> (unit -> 'a) -> 'a

val report_if_take_time : int -> string -> (unit -> 'a) -> 'a

(* similar to profile_code but print some information during execution too *)
val profile_code2 : string -> (unit -> 'a) -> 'a
```

3.4 Testing

[1]

```
(* Better than quickcheck, cos cant do a test_all_prop in haskell cos
 * prop were functions, whereas here we have not prop_Unix x = ... but
 * laws "unit" ...
 *
 * How to do without overloading ? objet ? can pass a generator as a
 * parameter, mais lourd, prefer automatic inferring of the
 * generator? But at the same time quickcheck does not do better cos
 * we must explictly type the property. So between a
 *    prop_unit:: [Int] -> [Int] -> bool ...
 *    prop_unit x = reverse [x] == [x]
 * and
 *    let _ = laws "unit" (fun x -> reverse [x] = [x]) (listg intg)
 * there is no real differences.
 *
 * Yes I define typeg generator but quickcheck too, he must define
 * class instance. I emulate the context Gen a => Gen [a] by making
 * listg take as a param a type generator. Moreover I have not the pb of
 * monad. I can do random independently, so my code is more simple
 * I think than the haskell code of quickcheck.
 *
```

[1]TODO: `unit test, no in ocaml, not needed, just use equal`

```
 * update: apparently Jane Street have copied some of my code for their
 * Ounit_util.ml and quichcheck.ml in their Core library :)
 *)


(*
let b = laws "unit" (fun x       -> reverse [x]          = [x]                   )ig
let b = laws "app " (fun (xs,ys) -> reverse (xs++ys)     = reverse ys++reverse xs)(pg (l
let b = laws "rev " (fun xs      -> reverse (reverse xs) = xs                    )(lg ig
let b = laws "appb" (fun (xs,ys) -> reverse (xs++ys)     = reverse xs++reverse ys)(pg (l
let b = laws "max"  (fun (x,y)   -> x <= y ==> (max x y  = y)                       )(pg

let b = laws2 "max"  (fun (x,y)   -> ((x <= y ==> (max x y  = y)), x <= y))(pg ig ig)
*)
```

19 ⟨*common.mli basic features* 14⟩+≡

```
  (*****************************************************************************)
  (* Test *)
  (*****************************************************************************)

  val example : bool -> unit
  (* generate failwith <string> when pb *)
  val example2 : string -> bool -> unit
  (* use Dumper to report when pb *)
  val assert_equal : 'a -> 'a -> unit

  val _list_bool : (string * bool) list ref
  val example3 : string -> bool -> unit
  val test_all : unit -> unit


  (* regression testing *)
  type score_result = Ok | Pb of string
  type score =      (string (* usually a filename *), score_result) Hashtbl.t
  type score_list = (string (* usually a filename *) * score_result) list
  val empty_score : unit -> score
  val regression_testing :
    score -> filename (* old score file on disk (usually in /tmp) *) -> unit
  val regression_testing_vs: score -> score -> score
  val total_scores : score -> int (* good *) * int (* total *)
  val print_score : score -> unit
  val print_total_score: score -> unit
```

```
(* quickcheck spirit *)
type 'a gen = unit -> 'a

(* quickcheck random generators *)
val ig : int gen
val lg : 'a gen -> 'a list gen
val pg : 'a gen -> 'b gen -> ('a * 'b) gen
val polyg : int gen
val ng : string gen

val oneofl : 'a list -> 'a gen
val oneof : 'a gen list -> 'a gen
val always : 'a -> 'a gen
val frequency : (int * 'a gen) list -> 'a gen
val frequencyl : (int * 'a) list -> 'a gen

val laws : string -> ('a -> bool) -> 'a gen -> 'a option

(* example of use:
 * let b = laws "unit" (fun x -> reverse [x] = [x])    ig
 *)

val statistic_number : 'a list -> (int * 'a) list
val statistic : 'a list -> (int * 'a) list

val laws2 :
  string -> ('a -> bool * 'b) -> 'a gen -> 'a option * (int * 'b) list
```

3.5 Persitence

20 ⟨*common.mli basic features* 14⟩+≡

```
(*****************************************************************************)
(* Persistence *)
(*****************************************************************************)

(* just wrappers around Marshal *)
val get_value : filename -> 'a
val read_value : filename -> 'a (* alias *)
val write_value : 'a -> filename -> unit
val write_back : ('a -> 'b) -> filename -> unit

(* wrappers that also use profile_code *)
val marshal__to_string:   'a -> Marshal.extern_flags list -> string
val marshal__from_string:   string -> int -> 'a
```

3.6 Counter

21a ⟨*common.mli basic features* 14⟩+≡
```
(*****************************************************************************)
(* Counter *)
(*****************************************************************************)
val _counter : int ref
val _counter2 : int ref
val _counter3 : int ref

val counter : unit -> int
val counter2 : unit -> int
val counter3 : unit -> int

type timestamp = int
```

3.7 Stringof

21b ⟨*common.mli basic features* 14⟩+≡
```
(*****************************************************************************)
(* String_of and (pretty) printing *)
(*****************************************************************************)

val string_of_string : (string -> string) -> string
val string_of_list : ('a -> string) -> 'a list -> string
val string_of_unit : unit -> string
val string_of_array : ('a -> string) -> 'a array -> string
val string_of_option : ('a -> string) -> 'a option -> string

val print_bool : bool -> unit
val print_option : ('a -> 'b) -> 'a option -> unit
val print_list : ('a -> 'b) -> 'a list -> unit
val print_between : (unit -> unit) -> ('a -> unit) -> 'a list -> unit

(* use Format internally *)
val pp_do_in_box : (unit -> unit) -> unit
val pp_f_in_box : (unit -> 'a) -> 'a
val pp_do_in_zero_box : (unit -> unit) -> unit
val pp : string -> unit

(* convert something printed using Format to print into a string *)
val format_to_string : (unit -> unit) (* printer *) -> string

(* works with _tab_level_print enabling to mix some calls to pp, pr2
 * and indent_do to sometimes use advanced indentation pretty printing
 * (with the pp* functions) and sometimes explicit and simple indendation
 * printing (with pr* and indent_do) *)
val adjust_pp_with_indent : (unit -> unit) -> unit
val adjust_pp_with_indent_and_header : string -> (unit -> unit) -> unit


val mk_str_func_of_assoc_conv:
  ('a * string) list -> (string -> 'a) * ('a -> string)
```

3.8 Macro

22a ⟨*common.mli basic features* 14⟩+≡

```
(*****************************************************************************)
(* Macro *)
(*****************************************************************************)

(* was working with my macro.ml4 *)
val macro_expand : string -> unit
```

```
(* I like the obj.func object notation. In OCaml cant use '.' so I use +>
 *
 * update: it seems that F# agrees with me :) but they use |>
 *)
```

3.9 Composition and control

22b ⟨*common.mli basic features* 14⟩+≡

```
(*****************************************************************************)
(* Composition/Control *)
(*****************************************************************************)

val ( +> ) : 'a -> ('a -> 'b) -> 'b
val ( +!> ) : 'a ref -> ('a -> 'a) -> unit
val ( $ ) : ('a -> 'b) -> ('b -> 'c) -> 'a -> 'c

val compose : ('a -> 'b) -> ('c -> 'a) -> 'c -> 'b
val flip : ('a -> 'b -> 'c) -> 'b -> 'a -> 'c

val curry : ('a * 'b -> 'c) -> 'a -> 'b -> 'c
val uncurry : ('a -> 'b -> 'c) -> 'a * 'b -> 'c

val id : 'a -> 'a
val do_nothing : unit -> unit
val forever : (unit -> unit) -> unit

val applyn : int -> ('a -> 'a) -> 'a -> 'a

class ['a] shared_variable_hook :
  'a ->
  object
    val mutable data : 'a
    val mutable registered : (unit -> unit) list
    method get : 'a
    method modify : ('a -> 'a) -> unit
    method register : (unit -> unit) -> unit
    method set : 'a -> unit
  end

val fixpoint : ('a -> 'a) -> 'a -> 'a
val fixpoint_for_object : ((< equal : 'a -> bool; .. > as 'a) -> 'a) -> 'a -> 'a

val add_hook : ('a -> ('a -> 'b) -> 'b) ref -> ('a -> ('a -> 'b) -> 'b) -> unit
val add_hook_action : ('a -> unit) ->   ('a -> unit) list ref -> unit
val run_hooks_action : 'a -> ('a -> unit) list ref -> unit

type 'a mylazy = (unit -> 'a)

(* emacs spirit *)
val save_excursion : 'a ref -> (unit -> 'b) -> 'b
val save_excursion_and_disable : bool ref -> (unit -> 'b) -> 'b
val save_excursion_and_enable :  bool ref -> (unit -> 'b) -> 'b

(* emacs spirit *)
val unwind_protect : (unit -> 'a) -> (exn -> 'b) -> 'a

(* java spirit *)
val finalize :       (unit -> 'a) -> (unit -> 'b) -> 'a

val memoized : ('a, 'b) Hashtbl.t -> 'a -> (unit -> 'b) -> 'b

val cache_in_ref : 'a option ref -> (unit -> 'a) -> 'a


(* take file from which computation is done, an extension, and the function
 * and will compute the function only once and then save result in
 * file ^ extension
 *)
val cache_computation :
  ?verbose:bool -> ?use_cache:bool -> filename  -> string (* extension *) ->
  (unit -> 'a) -> 'a

(* a more robust version where the client describes the dependencies of the
 * computation so it will relaunch the computation in 'f' if needed.
 *)
val cache_computation_robust :
  filename ->
  string (* extension for marshalled object *) ->
  (filename list * 'x) ->
  string (* extension for marshalled dependencies *) ->
  (unit -> 'a) ->
  'a



val once : ('a -> unit) -> ('a -> unit)

val before_leaving : ('a -> unit) -> 'a -> 'a

(* do some finalize, signal handling, unix exit conversion, etc *)
val main_boilerplate : (unit -> unit) -> unit


(* cf also the timeout function below that are control related too *)
```

3.10 Concurrency

24a ⟨*common.mli basic features* 14⟩+≡

```
(*****************************************************************************)
(* Concurrency *)
(*****************************************************************************)

(* how ensure really atomic file creation ? hehe :) *)
exception FileAlreadyLocked
val acquire_file_lock : filename -> unit
val release_file_lock : filename -> unit
```

3.11 Error management

24b ⟨*common.mli basic features* 14⟩+≡

```
(*****************************************************************************)
(* Error managment *)
(*****************************************************************************)
exception Todo
exception Impossible
exception Here
exception ReturnExn

exception Multi_found

exception WrongFormat of string


val internal_error : string -> 'a
val myassert : bool -> unit
val warning : string -> 'a -> 'a
val error_cant_have : 'a -> 'b

val exn_to_s : exn -> string
(* alias *)
val string_of_exn : exn -> string

type error = Error of string

type evotype = unit
val evoval : evotype
```

3.12 Environment

25 ⟨*common.mli basic features* 14⟩+≡

```
(*****************************************************************************)
(* Environment *)
(*****************************************************************************)

val check_stack_size: int -> unit
val check_stack_nbfiles: int -> unit

(* internally common.ml set Gc. parameters *)
val _init_gc_stack : unit

* Why define wrappers ? Arg not good enough ? Well the Arg.Rest is not that
* good and I need a way sometimes to get a list of argument.
*
* I could define maybe a new Arg.spec such as
* | String_list of (string list -> unit), but the action may require
* some flags to be set, so better to process this after all flags have
* been set by parse_options. So have to split. Otherwise it would impose
* an order of the options such as
* -verbose_parsing -parse_c file1 file2. and I really like to use bash
* history and add just at the end of my command a -profile for instance.
*
*
* Why want a -action arg1 arg2 arg3 ? (which in turn requires this
* convulated scheme ...) Why not use Arg.String action such as
* "-parse_c", Arg.String (fun file -> ...) ?
* I want something that looks like ocaml function but at the UNIX
* command line level. So natural to have this scheme instead of
* -taxo_file arg2 -sample_file arg3 -parse_c arg1.
*
*
* Why not use the toplevel ?
* - because to debug, ocamldebug is far superior to the toplevel
*   (can go back, can go directly to a specific point, etc).
*   I want a kind of testing at cmdline level.
* - Also I don't have file completion when in the ocaml toplevel.
*   I have to type "/path/to/xxx" without help.
*
*
* Why having variable flags ? Why use 'if !verbose_parsing then ...' ?
* why not use strings and do stuff like the following
* 'if (get_config "verbose_parsing") then ...'
* Because I want to make the interface for flags easier for the code
* that use it. The programmer should not be bothered wether this
* flag is set via args cmd line or a config file, so I want to make it
* as simple as possible, just use a global plain caml ref variable.
*
* Same spirit a little for the action. Instead of having function such as
* test_parsing_c, I could do it only via string. But I still prefer
* to have plain caml test functions. Also it makes it easier to call
* those functions from a toplevel for people who prefer the toplevel.
*
*
* So have flag_spec and action_spec. And in flag have debug_xxx flags,
* verbose_xxx flags and other flags.
*
* I would like to not have to separate the -xxx actions spec from the
* corresponding actions, but those actions may need more than one argument
* and so have to wait for parse_options, which in turn need the options
* spec, so circle.
*
* Also I dont want to mix code with data structures, so it's better that the
* options variable contain just a few stuff and have no side effects except
* setting global variables.
*
* Why not have a global variable such as Common.actions that
```

```
 * other modules modify ? No, I prefer to do less stuff behind programmer's
 * back so better to let the user merge the different options at call
 * site, but at least make it easier by providing shortcut for set of options.
 *
 *
 *


(* kind of unit testing framework, or toplevel like functionnality
 * at shell command line. I realize than in fact It follows a current trend
 * to have a main cmdline program where can then select different actions,
 * as in cvs/hg/git where do  hg <action> <arguments>, and the shell even
 * use a curried syntax :)
 *
 *
 * Not-perfect-but-basic-feels-right: an action
 * spec looks like this:
 *
 *    let actions () = [
 *      "-parse_taxo", "   <file>",
 *      Common.mk_action_1_arg test_parse_taxo;
 *      ...
 *     ]
 *
 * Not-perfect-but-basic-feels-right because for such functionality we
 * need a way to transform a string into a caml function and pass arguments
 * and the preceding design does exactly that, even if then the
 * functions that use this design are not so convenient to use (there
 * are 2 places where we need to pass those data, in the options and in the
 * main dispatcher).
 *
 * Also it's not too much intrusive. Still have an
 * action ref variable in the main.ml and can still use the previous
 * simpler way to do where the match args with in main.ml do the
 * dispatch.
 *
 * Use like this at option place:
 *   (Common.options_of_actions actionref (Test_parsing_c.actions())) ++
 * Use like this at dispatch action place:
 *   | xs when List.mem !action (Common.action_list all_actions) ->
 *        Common.do_action !action xs all_actions
 *
 *)
```

3.13 Arguments

28 ⟨*common.mli basic features* 14⟩+≡
```
(*****************************************************************************)
(* Arguments and command line *)
(*****************************************************************************)

type arg_spec_full = Arg.key * Arg.spec * Arg.doc
type cmdline_options = arg_spec_full list


type options_with_title = string * string * arg_spec_full list
type cmdline_sections = options_with_title list


(* A wrapper around Arg modules that have more logical argument order,
 * and returns the remaining args.
 *)
val parse_options :
  cmdline_options -> Arg.usage_msg -> string array -> string list

(* Another wrapper that does Arg.align automatically *)
val usage : Arg.usage_msg -> cmdline_options -> unit



(* Work with the options_with_title type way to organize a long
 * list of command line switches.
 *)
val short_usage :
  Arg.usage_msg -> short_opt:cmdline_options -> unit
val long_usage :
  Arg.usage_msg -> short_opt:cmdline_options -> long_opt:cmdline_sections ->
  unit

(* With the options_with_title way, we don't want the default -help and --help
 * so need adapter of Arg module, not just wrapper.
 *)
val arg_align2 : cmdline_options -> cmdline_options
val arg_parse2 :
  cmdline_options -> Arg.usage_msg -> (unit -> unit) (* short_usage func *) ->
  string list
```

```
(* The action lib. Useful to debug supart of your system. cf some of
 * my main.ml for example of use. *)
type flag_spec   = Arg.key * Arg.spec * Arg.doc
type action_spec = Arg.key * Arg.doc * action_func
   and action_func = (string list -> unit)

type cmdline_actions = action_spec list
exception WrongNumberOfArguments

val mk_action_0_arg : (unit -> unit)                        -> action_func
val mk_action_1_arg : (string -> unit)                      -> action_func
val mk_action_2_arg : (string -> string -> unit)            -> action_func
val mk_action_3_arg : (string -> string -> string -> unit) -> action_func

val mk_action_n_arg : (string list -> unit) -> action_func

val options_of_actions:
  string ref (* the action ref *) -> cmdline_actions -> cmdline_options
val action_list:
  cmdline_actions -> Arg.key list
val do_action:
  Arg.key -> string list (* args *) -> cmdline_actions -> unit
```

3.14 Equality

29 ⟨*common.mli basic features* 14⟩+≡

```
(*****************************************************************************)
(* Equality *)
(*****************************************************************************)

(* Using the generic (=) is tempting, but it backfires, so better avoid it *)

(* To infer all the code that use an equal, and that should be
 * transformed, is not that easy, because (=) is used by many
 * functions, such as List.find, List.mem, and so on. The strategy to find
 * them is to turn what you were previously using into a function, because
 * (=) return an exception when applied to a function, then you simply
 * use ocamldebug to detect where the code has to be transformed by
 * finding where the exception was launched from.
 *)

val (=|=) : int    -> int    -> bool
val (=<=) : char   -> char   -> bool
val (=$=) : string -> string -> bool
val (=:=) : bool   -> bool   -> bool

(* the evil generic (=). I define another symbol to more easily detect
 * it, cos the '=' sign is syntaxically overloaded in caml. It is also
 * used to define function.
 *)
val (=*=): 'a -> 'a -> bool

(* if want to restrict the use of '=', uncomment this:
 *
 * val (=): unit -> unit -> bool
 *
 * But it will not forbid you to use caml functions like List.find, List.mem
 * which internaly use this convenient but evolution-unfriendly (=)
*)
```

Chapter 4

Basic types

4.1 Bool

31a ⟨*common.mli for basic types* 31a⟩≡
```
(*****************************************************************************)
(* Bool *)
(*****************************************************************************)

val ( ||| ) : 'a -> 'a -> 'a
val ( ==> ) : bool -> bool -> bool
val xor : 'a -> 'a -> bool
```

4.2 Char

31b ⟨*common.mli for basic types* 31a⟩+≡
```
(*****************************************************************************)
(* Char *)
(*****************************************************************************)

val string_of_char : char -> string
val string_of_chars : char list -> string

val is_single : char -> bool
val is_symbol : char -> bool
val is_space : char -> bool
val is_upper : char -> bool
val is_lower : char -> bool
val is_alpha : char -> bool
val is_digit : char -> bool
val cbetween : char -> char -> char -> bool
```

4.3 Num

32 ⟨*common.mli for basic types* 31a⟩+≡

```
(*****************************************************************************)
(* Num *)
(*****************************************************************************)

val ( /! ) : int -> int -> int

val do_n : int -> (unit -> unit) -> unit
val foldn : ('a -> int -> 'a) -> 'a -> int -> 'a

val pi : float
val pi2 : float
val pi4 : float

val deg_to_rad : float -> float

val clampf : float -> float

val square : float -> float
val power : int -> int -> int

val between : 'a -> 'a -> 'a -> bool
val between_strict : int -> int -> int -> bool
val bitrange : int -> int -> bool

val prime1 : int -> int option
val prime : int -> int option

val sum : int list -> int
val product : int list -> int

val decompose : int -> int list

val mysquare : int -> int
val sqr : float -> float

type compare = Equal | Inf | Sup
val ( <=> ) : 'a -> 'a -> compare
val ( <==> ) : 'a -> 'a -> int

type uint = int
```

```
val int_of_stringchar : string -> int
val int_of_base : string -> int -> int
val int_of_stringbits : string -> int
val int_of_octal : string -> int
val int_of_all : string -> int

(* useful but sometimes when want grep for all places where do modif,
 * easier to have just code using ':=' and '<-' to do some modifications.
 * In the same way avoid using {contents = xxx} to build some ref.
 *)
val ( += ) : int ref -> int -> unit
val ( -= ) : int ref -> int -> unit

val pourcent: int -> int -> int
val pourcent_float: int -> int -> float
val pourcent_float_of_floats: float -> float -> float

val pourcent_good_bad: int -> int -> int
val pourcent_good_bad_float: int -> int -> float

type 'a max_with_elem = int ref * 'a ref
val update_max_with_elem:
  'a max_with_elem -> is_better:(int -> int ref -> bool) -> int * 'a -> unit
```

33 ⟨*common.mli for basic types* 31a⟩+≡

```
(*****************************************************************************)
(* Numeric/overloading *)
(*****************************************************************************)

type 'a numdict =
    NumDict of
      (('a -> 'a -> 'a) * ('a -> 'a -> 'a) * ('a -> 'a -> 'a) * ('a -> 'a))
val add : 'a numdict -> 'a -> 'a -> 'a
val mul : 'a numdict -> 'a -> 'a -> 'a
val div : 'a numdict -> 'a -> 'a -> 'a
val neg : 'a numdict -> 'a -> 'a

val numd_int   : int   numdict
val numd_float : float numdict

val testd : 'a numdict -> 'a -> 'a


module ArithFloatInfix : sig
```

```
    val (+) : float -> float -> float
    val (-) : float -> float -> float
    val (/) : float -> float -> float
    val ( * ) : float -> float -> float


    val (+..) : int -> int -> int
    val (-..) : int -> int -> int
    val (/..) : int -> int -> int
    val ( *..) : int -> int -> int

    val (+=) : float ref -> float -> unit
end
```

4.4 Random

34a ⟨*common.mli for basic types* 31a⟩+≡

```
(*****************************************************************************)
(* Random *)
(*****************************************************************************)

val _init_random : unit
val random_list : 'a list -> 'a
val randomize_list : 'a list -> 'a list
val random_subset_of_list : int -> 'a list -> 'a list
```

4.5 Tuples

34b ⟨*common.mli for basic types* 31a⟩+≡

```
(*****************************************************************************)
(* Tuples *)
(*****************************************************************************)

type 'a pair = 'a * 'a
type 'a triple = 'a * 'a * 'a

val fst3 : 'a * 'b * 'c -> 'a
val snd3 : 'a * 'b * 'c -> 'b
val thd3 : 'a * 'b * 'c -> 'c

val sndthd : 'a * 'b * 'c -> 'b * 'c

val map_fst : ('a -> 'b) -> 'a * 'c -> 'b * 'c
val map_snd : ('a -> 'b) -> 'c * 'a -> 'c * 'b
val pair : ('a -> 'b) -> 'a * 'a -> 'b * 'b

val snd : 'a * 'b -> 'b (* alias *)
val fst : 'a * 'b -> 'a (* alias *)

val double : 'a -> 'a * 'a
val swap : 'a * 'b -> 'b * 'a

(* maybe a sign of bad programming if use those functions :) *)
val tuple_of_list1 : 'a list -> 'a
val tuple_of_list2 : 'a list -> 'a * 'a
val tuple_of_list3 : 'a list -> 'a * 'a * 'a
val tuple_of_list4 : 'a list -> 'a * 'a * 'a * 'a
val tuple_of_list5 : 'a list -> 'a * 'a * 'a * 'a * 'a
val tuple_of_list6 : 'a list -> 'a * 'a * 'a * 'a * 'a * 'a
```

4.6 Maybe

35 ⟨*common.mli for basic types* 31a⟩+≡

```
(*****************************************************************************)
(* Maybe *)
(*****************************************************************************)

type ('a, 'b) either = Left of 'a | Right of 'b
type ('a, 'b, 'c) either3 = Left3 of 'a | Middle3 of 'b | Right3 of 'c

val just : 'a option -> 'a
val some : 'a option -> 'a (* alias *)

val fmap :       ('a -> 'b) -> 'a option -> 'b option
val map_option : ('a -> 'b) -> 'a option -> 'b option (* alias *)

val do_option : ('a -> unit) -> 'a option -> unit

val optionise : (unit -> 'a) -> 'a option

val some_or : 'a option -> 'a -> 'a

val partition_either :
  ('a -> ('b, 'c) either) -> 'a list -> 'b list * 'c list
val partition_either3 :
    ('a -> ('b, 'c, 'd) either3) -> 'a list -> 'b list * 'c list * 'd list

val filter_some : 'a option list -> 'a list
val map_filter : ('a -> 'b option) -> 'a list -> 'b list
val find_some : ('a -> 'b option) -> 'a list -> 'b

val list_to_single_or_exn: 'a list -> 'a

val while_some: gen:(unit-> 'a option) -> f:('a -> 'b) -> unit -> 'b list
```

4.7 TriBool

36a ⟨*common.mli for basic types* 31a⟩+≡

```
(*****************************************************************************)
(* TriBool *)
(*****************************************************************************)
type bool3 = True3 | False3 | TrueFalsePb3 of string
```

4.8 Strings

36b ⟨*common.mli for basic types* 31a⟩+≡

```
(*****************************************************************************)
(* Strings *)
(*****************************************************************************)

val slength : string -> int (* alias *)
val concat : string -> string list -> string (* alias *)

val i_to_s : int -> string
val s_to_i : string -> int

(* strings take space in memory. Better when can share the space used by
 * similar strings.
 *)
val _shareds : (string, string) Hashtbl.t
val shared_string : string -> string

val chop : string -> string
val chop_dirsymbol : string -> string

val ( <!!> ) : string -> int * int -> string
val ( <!> ) : string -> int -> char

val take_string: int -> string -> string
val take_string_safe: int -> string -> string
```

```
 val split_on_char : char -> string -> string list

 val lowercase : string -> string

 val quote : string -> string

 val null_string : string -> bool
 val is_blank_string : string -> bool
 val is_string_prefix : string -> string -> bool


 val plural : int -> string -> string

 val showCodeHex : int list -> unit

 val size_mo_ko : int -> string
 val size_ko : int -> string

 val edit_distance: string -> string -> int

 val md5sum_of_string : string -> string

 val wrap: ?width:int -> string -> string


(* Note: OCaml Str regexps are different from Perl regexp:
 *  - The OCaml regexp must match the entire way.
 *    So  "testBee" =~ "Bee" is wrong
 *    but "testBee" =~ ".*Bee" is right
 *    Can have the perl behavior if use  Str.search_forward instead of
 *    Str.string_match.
 *  - Must add some additional \ in front of some special char. So use
 *    \\( \\|  and also \\b
 *  - It does not always handle newlines very well.
 *  - \\b does consider _ but not numbers in indentifiers.
 *
 * Note: PCRE regexps are then different from Str regexps ...
 *  - just use '(' ')' for grouping, not '\\)'
 *  - still need \\b for word boundary, but this time it works ...
 *    so can match some word that have some digits in them.
 *
 *)
```

4.9 Regexp

38 ⟨*common.mli for basic types* 31a⟩+≡
```
(*****************************************************************************)
(* Regexp *)
(*****************************************************************************)

val regexp_alpha : Str.regexp
val regexp_word : Str.regexp

val _memo_compiled_regexp : (string, Str.regexp) Hashtbl.t
val ( =~ ) : string -> string -> bool
val ( ==~ ) : string -> Str.regexp -> bool



val regexp_match : string -> string -> string

val matched : int -> string -> string

(* not yet politypic functions in ocaml *)
val matched1 : string -> string
val matched2 : string -> string * string
val matched3 : string -> string * string * string
val matched4 : string -> string * string * string * string
val matched5 : string -> string * string * string * string * string
val matched6 : string -> string * string * string * string * string * string
val matched7 : string -> string * string * string * string * string * string * string

val string_match_substring : Str.regexp -> string -> bool

val split : string (* sep regexp *) -> string -> string list
val join : string (* sep *) -> string list -> string

val split_list_regexp : string -> string list -> (string * string list) list

val all_match : string (* regexp *) -> string -> string list
val global_replace_regexp :
  string (* regexp *) -> (string -> string) -> string -> string

val regular_words: string -> string list
val contain_regular_word: string -> bool
```

4.10 Filenames

39a ⟨*common.mli for basic types* 31a⟩+≡

```
(*****************************************************************************)
(* Filenames *)
(*****************************************************************************)

(* now at beginning of this file: type filename = string *)
val dirname : string -> string
val basename : string -> string

val filesuffix : filename -> string
val fileprefix : filename -> string

val adjust_ext_if_needed : filename -> string -> filename

(* db for dir, base *)
val db_of_filename : filename -> (string * filename)
val filename_of_db : (string * filename) -> filename

(* dbe for dir, base, ext *)
val dbe_of_filename : filename -> string * string * string
val dbe_of_filename_nodot : filename -> string * string * string
(* Left (d,b,e) | Right (d,b)  if file has no extension *)
val dbe_of_filename_safe :
  filename -> (string * string * string,  string * string) either

val filename_of_dbe : string * string * string -> filename

(* ex: replace_ext "toto.c" "c" "var" *)
val replace_ext: filename -> string -> string -> filename

(* remove the ., .. *)
val normalize_path : filename -> filename

val relative_to_absolute : filename -> filename

val is_relative: filename -> bool
val is_absolute: filename -> bool

val filename_without_leading_path : string -> filename -> filename
```

4.11 i18n

39b ⟨*common.mli for basic types* 31a⟩+≡

```
(*****************************************************************************)
(* i18n *)
(*****************************************************************************)
type langage =
  | English
  | Francais
  | Deutsch
```

4.12 Date

40 ⟨*common.mli for basic types* 31a⟩+≡
```
(*****************************************************************************)
(* Dates *)
(*****************************************************************************)

(* can also use ocamlcalendar, but heavier, use many modules ... *)

type month =
  | Jan  | Feb  | Mar  | Apr  | May  | Jun
  | Jul  | Aug  | Sep  | Oct  | Nov  | Dec
type year = Year of int
type day = Day of int

type date_dmy = DMY of day * month * year

type hour = Hour of int
type minute = Min of int
type second = Sec of int

type time_hms = HMS of hour * minute * second

type full_date = date_dmy * time_hms


(* intervalle *)
type days = Days of int

type time_dmy = TimeDMY of day * month * year


(* from Unix *)
type float_time = float


val mk_date_dmy : int -> int -> int -> date_dmy
val check_date_dmy : date_dmy -> unit
val check_time_dmy : time_dmy -> unit
val check_time_hms : time_hms -> unit

val int_to_month : int -> string
val int_of_month : month -> int
val month_of_string : string -> month
val month_of_string_long : string -> month
val string_of_month : month -> string

val string_of_date_dmy : date_dmy -> string
val date_dmy_of_string : string -> date_dmy

val string_of_unix_time : ?langage:langage -> Unix.tm -> string
val short_string_of_unix_time : ?langage:langage -> Unix.tm -> string
val string_of_floattime: ?langage:langage -> float_time -> string
val short_string_of_floattime: ?langage:langage -> float_time -> string

val floattime_of_string: string -> float_time

val dmy_to_unixtime: date_dmy -> float_time * Unix.tm
val unixtime_to_dmy: Unix.tm -> date_dmy
val unixtime_to_floattime: Unix.tm -> float_time
val floattime_to_unixtime: float_time -> Unix.tm

val sec_to_days : int -> string
val sec_to_hours : int -> string

val today : unit -> float_time
val yesterday : unit -> float_time
val tomorrow : unit -> float_time

val lastweek : unit -> float_time
val lastmonth : unit -> float_time

val week_before: float_time -> float_time
val month_before: float_time -> float_time
val week_after: float_time -> float_time

val days_in_week_of_day : float_time -> float_time list

val first_day_in_week_of_day : float_time -> float_time
val last_day_in_week_of_day : float_time -> float_time
val day_secs: float_time

val rough_days_since_jesus : date_dmy -> days
val rough_days_between_dates : date_dmy -> date_dmy -> days

val string_of_unix_time_lfs : Unix.tm -> string

val is_more_recent : date_dmy -> date_dmy -> bool
val max_dmy : date_dmy -> date_dmy -> date_dmy
val min_dmy : date_dmy -> date_dmy -> date_dmy
val maximum_dmy : date_dmy list -> date_dmy
val minimum_dmy : date_dmy list -> date_dmy
```

4.13 Lines/words/strings

42a ⟨*common.mli for basic types* 31a⟩+≡

```
(*****************************************************************************)
(* Lines/Words/Strings *)
(*****************************************************************************)

val list_of_string : string -> char list

val lines : string -> string list
val unlines : string list -> string

val words : string -> string list
val unwords : string list -> string

val split_space : string -> string list

val lines_with_nl : string -> string list

val nblines : string -> int
val words_of_string_with_newlines: string -> string list
```

4.14 Process/files

42b ⟨*common.mli for basic types* 31a⟩+≡

```
(*****************************************************************************)
(* Process/Files *)
(*****************************************************************************)
val cat :      filename -> string list
val cat_orig : filename -> string list
val cat_array: filename -> string array

val uncat: string list -> filename -> unit

val interpolate : string -> string list

val echo : string -> string

val usleep : int -> unit

val process_output_to_list : string -> string list
val cmd_to_list :            string -> string list (* alias *)
val cmd_to_list_and_status : string -> string list * Unix.process_status

val command2 : string -> unit
val _batch_mode: bool ref

val y_or_no: string -> bool

val command2_y_or_no : string -> bool
val command2_y_or_no_exit_if_no : string -> unit

val do_in_fork : (unit -> unit) -> int

val mkdir: ?mode:Unix.file_perm -> string -> unit

val read_file : filename -> string
val write_file : file:filename -> string -> unit

val filesize : filename -> int
val filemtime : filename -> float

val nblines_file : filename -> int

val lfile_exists : filename -> bool
val is_directory : filename -> bool

val capsule_unix : ('a -> unit) -> 'a -> unit

val readdir_to_kind_list : string -> Unix.file_kind -> string list
val readdir_to_dir_list : string -> string list
val readdir_to_file_list : string -> string list
val readdir_to_link_list : string -> string list
val readdir_to_dir_size_list : string -> (string * int) list

val glob : string -> filename list
val files_of_dir_or_files :
  string (* ext *) -> string list -> filename list
val files_of_dir_or_files_no_vcs :
  string (* ext *) -> string list -> filename list
(* use a post filter =~ for the ext filtering *)
val files_of_dir_or_files_no_vcs_post_filter :
  string (* regexp *) -> string list -> filename list
val files_of_dir_or_files_no_vcs_nofilter:
 string list -> filename list

val sanity_check_files_and_adjust :
  string (* ext *) -> string list -> filename list


type rwx = [ `R | `W | `X ] list
val file_perm_of : u:rwx -> g:rwx -> o:rwx -> Unix.file_perm

val has_env : string -> bool

(* scheme spirit. do a finalize so no leak. *)
val with_open_outfile :
  filename -> ((string -> unit) * out_channel -> 'a) -> 'a
val with_open_infile :
  filename -> (in_channel -> 'a) -> 'a
val with_open_outfile_append :
  filename -> ((string -> unit) * out_channel -> 'a) -> 'a

val with_open_stringbuf :
  (((string -> unit) * Buffer.t) -> unit) -> string

exception Timeout

(* subtil: have to make sure that Timeout is not intercepted before here. So
 * avoid exn handler such as try (...) with _ -> cos Timeout will not bubble up
 * enough. In such case, add a case before such as
 * with Timeout -> raise Timeout | _ -> ...
 *
 * The same is true for UnixExit (see below).
 *)
val timeout_function : int -> (unit -> 'a) -> 'a

val timeout_function_opt : int option -> (unit -> 'a) -> 'a


(* creation of /tmp files, a la gcc
 * ex: new_temp_file "cocci" ".c" will give "/tmp/cocci-3252-434465.c"
 *)
val _temp_files_created : string list ref
(* see flag: val save_tmp_files : bool ref *)
val new_temp_file : string (* prefix *) -> string (* suffix *) -> filename
val erase_temp_files : unit -> unit
val erase_this_temp_file : filename -> unit

(* If the user use some exit 0 in his code, then no one can intercept this
 * exit and do something before exiting. There is exn handler for exit 0
 * so better never use exit 0 but instead use an exception and just at
 * the very toplevel transform this exn in a unix exit code.
 *
 * subtil: same problem than with Timeout. Do not intercept such exception
 * with some blind try (...) with _ -> ...
 *)
exception UnixExit of int
val exn_to_real_unixexit : (unit -> 'a) -> 'a
```

Chapter 5

Collection

1

5.1 List

46 ⟨*common.mli for collection types* 46⟩≡
```
(*****************************************************************************)
(* List *)
(*****************************************************************************)


(* tail recursive efficient map (but that also reverse the element!) *)
val map_eff_rev : ('a -> 'b) -> 'a list -> 'b list
(* tail recursive efficient map, use accumulator  *)
val acc_map : ('a -> 'b) -> 'a list -> 'b list


val zip : 'a list -> 'b list -> ('a * 'b) list
val zip_safe : 'a list -> 'b list -> ('a * 'b) list
val unzip : ('a * 'b) list -> 'a list * 'b list

val take : int -> 'a list -> 'a list
val take_safe : int -> 'a list -> 'a list
val take_until : ('a -> bool) -> 'a list -> 'a list
val take_while : ('a -> bool) -> 'a list -> 'a list

val drop : int -> 'a list -> 'a list
val drop_while : ('a -> bool) -> 'a list -> 'a list
val drop_until : ('a -> bool) -> 'a list -> 'a list
```

[1]TODO: `See also ocollection.mli`

```
val span : ('a -> bool) -> 'a list -> 'a list * 'a list

val skip_until : ('a list -> bool) -> 'a list -> 'a list
val skipfirst : (* Eq a *) 'a -> 'a list -> 'a list

(* cf also List.partition *)
val fpartition : ('a -> 'b option) -> 'a list -> 'b list * 'a list

val groupBy : ('a -> 'a -> bool) -> 'a list -> 'a list list
val exclude_but_keep_attached: ('a -> bool) -> 'a list -> ('a * 'a list) list
val group_by_post: ('a -> bool) -> 'a list -> ('a list * 'a) list *    'a list
val group_by_pre:  ('a -> bool) -> 'a list -> 'a list *    ('a * 'a list) list
val group_by_mapped_key: ('a -> 'b) -> 'a list -> ('b * 'a list) list

(* Use hash internally to not be in O(n2). If you want to use it on a
 * simple list, then first do a List.map to generate a key, for instance the
 * first char of the element, and then use this function.
 *)
val group_assoc_bykey_eff : ('a * 'b) list -> ('a * 'b list) list

val splitAt : int -> 'a list -> 'a list * 'a list

val split_when: ('a -> bool) -> 'a list -> 'a list * 'a * 'a list
val split_gen_when: ('a list -> 'a list option) -> 'a list -> 'a list list

val pack : int -> 'a list -> 'a list list


val enum : int -> int -> int list
val repeat : 'a -> int -> 'a list
val generate : int -> 'a -> 'a list



val index_list   : 'a list -> ('a * int) list
val index_list_1 : 'a list -> ('a * int) list
val index_list_and_total   : 'a list -> ('a * int * int) list

val iter_index : ('a -> int -> 'b) -> 'a list -> unit
val map_index : ('a -> int -> 'b) -> 'a list -> 'b list
val filter_index : (int -> 'a -> bool) -> 'a list -> 'a list
val fold_left_with_index : ('a -> 'b -> int -> 'a) -> 'a -> 'b list -> 'a

val nth : 'a list -> int -> 'a
val rang : (* Eq a *) 'a -> 'a list -> int
val last_n : int -> 'a list -> 'a list



val snoc : 'a -> 'a list -> 'a list
val cons : 'a -> 'a list -> 'a list
val uncons : 'a list -> 'a * 'a list
val safe_tl : 'a list -> 'a list
val head_middle_tail : 'a list -> 'a * 'a list * 'a
val last : 'a list -> 'a
val list_init : 'a list -> 'a list
val list_last : 'a list -> 'a
val removelast : 'a list -> 'a list

val inits : 'a list -> 'a list list
val tails : 'a list -> 'a list list


val ( ++ ) : 'a list -> 'a list -> 'a list

val foldl1 : ('a -> 'a -> 'a) -> 'a list -> 'a
val fold_k : ('a -> 'b -> ('a -> 'a) -> 'a) -> ('a -> 'a) -> 'a -> 'b list -> 'a
val fold_right1 : ('a -> 'a -> 'a) -> 'a list -> 'a
val fold_left : ('a -> 'b -> 'a) -> 'a -> 'b list -> 'a

val rev_map : ('a -> 'b) -> 'a list -> 'b list

val join_gen : 'a -> 'a list -> 'a list

val do_withenv :
  (('a -> 'b) -> 'c -> 'd) -> ('e -> 'a -> 'b * 'e) -> 'e -> 'c -> 'd * 'e
val map_withenv : ('a -> 'b -> 'c * 'a) -> 'a -> 'b list -> 'c list * 'a
val map_withkeep: ('a -> 'b) -> 'a list -> ('b * 'a) list

val collect_accu : ('a -> 'b list) -> 'b list -> 'a list -> 'b list
val collect : ('a -> 'b list) -> 'a list -> 'b list

val remove : 'a -> 'a list -> 'a list

val exclude : ('a -> bool) -> 'a list -> 'a list

(* Not like unix uniq command line tool that only delete contiguous repeated
 * line. Here we delete any repeated line (here list element).
 *)
val uniq : 'a list -> 'a list
val uniq_eff: 'a list -> 'a list
val has_no_duplicate: 'a list -> bool
val is_set_as_list: 'a list -> bool
val get_duplicates: 'a list -> 'a list

val doublon : 'a list -> bool

val reverse : 'a list -> 'a list (* alias *)
val rev : 'a list -> 'a list (* alias *)
val rotate : 'a list -> 'a list

val map_flatten : ('a -> 'b list) -> 'a list -> 'b list

val map2 : ('a -> 'b) -> 'a list -> 'b list
val map3 : ('a -> 'b) -> 'a list -> 'b list


val maximum : 'a list -> 'a
val minimum : 'a list -> 'a

val most_recurring_element: 'a list -> 'a
val count_elements_sorted_highfirst: 'a list -> ('a * int) list

val min_with : ('a -> 'b) -> 'a list -> 'a
val two_mins_with : ('a -> 'b) -> 'a list -> 'a * 'a

val all_assoc : (* Eq a *) 'a -> ('a * 'b) list -> 'b list
val prepare_want_all_assoc : ('a * 'b) list -> ('a * 'b list) list

val or_list : bool list -> bool
val and_list : bool list -> bool

val sum_float : float list -> float
val sum_int : int list -> int
val avg_list: int list -> float

val return_when : ('a -> 'b option) -> 'a list -> 'b


val grep_with_previous : ('a -> 'a -> bool) -> 'a list -> 'a list
val iter_with_previous : ('a -> 'a -> 'b) -> 'a list -> unit

val iter_with_before_after :
 ('a list -> 'a -> 'a list -> unit) -> 'a list -> unit

val get_pair : 'a list -> ('a * 'a) list
val permutation : 'a list -> 'a list list

val remove_elem_pos : int -> 'a list -> 'a list
val insert_elem_pos : ('a * int) -> 'a list -> 'a list
val uncons_permut :   'a list -> (('a * int) * 'a list) list
val uncons_permut_lazy :   'a list -> (('a * int) * 'a list Lazy.t) list


val pack_sorted : ('a -> 'a -> bool) -> 'a list -> 'a list list

val keep_best : ('a * 'a -> 'a option) -> 'a list -> 'a list
val sorted_keep_best : ('a -> 'a -> 'a option) -> 'a list -> 'a list


val cartesian_product : 'a list -> 'b list -> ('a * 'b) list

(* old stuff *)
val surEnsemble : 'a list -> 'a list list -> 'a list list
val realCombinaison : 'a list -> 'a list list
val combinaison : 'a list -> ('a * 'a) list
val insere : 'a -> 'a list list -> 'a list list
val insereListeContenant : 'a list -> 'a -> 'a list list -> 'a list list
val fusionneListeContenant : 'a * 'a -> 'a list list -> 'a list list
```

5.2 Array

50a ⟨*common.mli for collection types* 46⟩+≡

```
(*****************************************************************************)
(* Arrays *)
(*****************************************************************************)

val array_find_index : (int -> bool) -> 'a array -> int
val array_find_index_via_elem : ('a -> bool) -> 'a array -> int

(* for better type checking, as sometimes when have an 'int array', can
 * easily mess up the index from the value.
 *)
type idx = Idx of int
val next_idx: idx -> idx
val int_of_idx: idx -> int

val array_find_index_typed : (idx -> bool) -> 'a array -> idx
```

50b ⟨*common.mli for collection types* 46⟩+≡

```
(*****************************************************************************)
(* Fast array *)
(*****************************************************************************)

(* ?? *)
```

5.3 Matrix

51a ⟨*common.mli for collection types* 46⟩+≡

```
(*****************************************************************************)
(* Matrix *)
(*****************************************************************************)

type 'a matrix = 'a array array

val map_matrix : ('a -> 'b) -> 'a matrix -> 'b matrix

val make_matrix_init:
  nrow:int -> ncolumn:int -> (int -> int -> 'a) -> 'a matrix

val iter_matrix:
  (int -> int -> 'a -> unit) -> 'a matrix -> unit

val nb_rows_matrix: 'a matrix -> int
val nb_columns_matrix: 'a matrix -> int

val rows_of_matrix: 'a matrix -> 'a list list
val columns_of_matrix: 'a matrix -> 'a list list

val all_elems_matrix_by_row: 'a matrix -> 'a list
```

5.4 Set

51b ⟨*common.mli for collection types* 46⟩+≡

```
(*****************************************************************************)
(* Set. But have a look too at set*.mli; it's better. Or use Hashtbl. *)
(*****************************************************************************)

type 'a set = 'a list

val empty_set : 'a set

val insert_set : 'a -> 'a set -> 'a set
val single_set : 'a -> 'a set
val set : 'a list -> 'a set

val is_set: 'a list -> bool

val exists_set : ('a -> bool) -> 'a set -> bool
val forall_set : ('a -> bool) -> 'a set -> bool

val filter_set : ('a -> bool) -> 'a set -> 'a set
val fold_set : ('a -> 'b -> 'a) -> 'a -> 'b set -> 'a
val map_set : ('a -> 'b) -> 'a set -> 'b set

val member_set : 'a -> 'a set -> bool
val find_set : ('a -> bool) -> 'a list -> 'a

val sort_set : ('a -> 'a -> int) -> 'a list -> 'a list

val iter_set : ('a -> unit) -> 'a list -> unit

val top_set : 'a set -> 'a

val inter_set : 'a set -> 'a set -> 'a set
val union_set : 'a set -> 'a set -> 'a set
val minus_set : 'a set -> 'a set -> 'a set

val union_all : ('a set) list -> 'a set

val big_union_set : ('a -> 'b set) -> 'a set -> 'b set
val card_set : 'a set -> int

val include_set : 'a set -> 'a set -> bool
val equal_set : 'a set -> 'a set -> bool
val include_set_strict : 'a set -> 'a set -> bool

(* could put them in Common.Infix *)
val ( $*$ ) : 'a set -> 'a set -> 'a set
val ( $+$ ) : 'a set -> 'a set -> 'a set
val ( $-$ ) : 'a set -> 'a set -> 'a set

val ( $?$ ) : 'a -> 'a set -> bool
val ( $<$ ) : 'a set -> 'a set -> bool
val ( $<=$ ) : 'a set -> 'a set -> bool
val ( $=$ ) : 'a set -> 'a set -> bool

val ( $@$ ) : 'a list -> 'a list -> 'a list

val nub : 'a list -> 'a list
```

```
(* use internally a hash and return
 * - the common part,
 * - part only in a,
 * - part only in b
 *)
val diff_two_say_set_eff : 'a list -> 'a list ->
  'a list * 'a list * 'a list
```

53a ⟨*common.mli for collection types* 46⟩+≡

```
(*****************************************************************************)
(* Set as normal list *)
(*****************************************************************************)

(* cf above *)
```

53b ⟨*common.mli for collection types* 46⟩+≡

```
(*****************************************************************************)
(* Set as sorted list *)
(*****************************************************************************)
```

53c ⟨*common.mli for collection types* 46⟩+≡

```
(*****************************************************************************)
(* Sets specialized *)
(*****************************************************************************)

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

5.5 Assoc

53d ⟨*common.mli for collection types* 46⟩+≡

```
(*****************************************************************************)
(* Assoc. But have a look too at Mapb.mli; it's better. Or use Hashtbl. *)
(*****************************************************************************)

type ('a, 'b) assoc = ('a * 'b) list

val assoc_to_function : (* Eq a *) ('a, 'b) assoc -> ('a -> 'b)

val empty_assoc : ('a, 'b) assoc
val fold_assoc : ('a -> 'b -> 'a) -> 'a -> 'b list -> 'a
val insert_assoc : 'a -> 'a list -> 'a list
val map_assoc : ('a -> 'b) -> 'a list -> 'b list
val filter_assoc : ('a -> bool) -> 'a list -> 'a list

val assoc : 'a -> ('a * 'b) list -> 'b

val keys : ('a * 'b) list -> 'a list
val lookup : 'a -> ('a * 'b) list -> 'b

val del_assoc : 'a -> ('a * 'b) list -> ('a * 'b) list
val replace_assoc : 'a * 'b -> ('a * 'b) list -> ('a * 'b) list
val apply_assoc : 'a -> ('b -> 'b) -> ('a * 'b) list -> ('a * 'b) list

val big_union_assoc : ('a -> 'b set) -> 'a list -> 'b set

val assoc_reverse : ('a * 'b) list -> ('b * 'a) list
val assoc_map : ('a * 'b) list -> ('a * 'b) list -> ('a * 'a) list

val lookup_list : 'a -> ('a, 'b) assoc list -> 'b
val lookup_list2 : 'a -> ('a, 'b) assoc list -> 'b * int

val assoc_option : 'a -> ('a, 'b) assoc -> 'b option
val assoc_with_err_msg : 'a -> ('a, 'b) assoc -> 'b

val sort_by_val_lowfirst: ('a,'b) assoc -> ('a * 'b) list
val sort_by_val_highfirst: ('a,'b) assoc -> ('a * 'b) list

val sort_by_key_lowfirst: (int,'b) assoc -> (int * 'b) list
val sort_by_key_highfirst: (int,'b) assoc -> (int * 'b) list

val sortgen_by_key_lowfirst: ('a,'b) assoc -> ('a * 'b) list
val sortgen_by_key_highfirst: ('a,'b) assoc -> ('a * 'b) list
```

54 ⟨*common.mli for collection types* 46⟩+≡

```
(*****************************************************************************)
(* Assoc, specialized. *)
(*****************************************************************************)

module IntMap :
  sig
    type key = int
    type +'a t
    val empty : 'a t
    val is_empty : 'a t -> bool
    val add : key -> 'a -> 'a t -> 'a t
    val find : key -> 'a t -> 'a
    val remove : key -> 'a t -> 'a t
    val mem : key -> 'a t -> bool
    val iter : (key -> 'a -> unit) -> 'a t -> unit
    val map : ('a -> 'b) -> 'a t -> 'b t
    val mapi : (key -> 'a -> 'b) -> 'a t -> 'b t
    val fold : (key -> 'a -> 'b -> 'b) -> 'a t -> 'b -> 'b
    val compare : ('a -> 'a -> int) -> 'a t -> 'a t -> int
    val equal : ('a -> 'a -> bool) -> 'a t -> 'a t -> bool
  end
val intmap_to_list : 'a IntMap.t -> (IntMap.key * 'a) list
val intmap_string_of_t : 'a -> 'b -> string

module IntIntMap :
  sig
    type key = int * int
    type +'a t
    val empty : 'a t
    val is_empty : 'a t -> bool
    val add : key -> 'a -> 'a t -> 'a t
    val find : key -> 'a t -> 'a
    val remove : key -> 'a t -> 'a t
    val mem : key -> 'a t -> bool
    val iter : (key -> 'a -> unit) -> 'a t -> unit
    val map : ('a -> 'b) -> 'a t -> 'b t
    val mapi : (key -> 'a -> 'b) -> 'a t -> 'b t
    val fold : (key -> 'a -> 'b -> 'b) -> 'a t -> 'b -> 'b
    val compare : ('a -> 'a -> int) -> 'a t -> 'a t -> int
    val equal : ('a -> 'a -> bool) -> 'a t -> 'a t -> bool
  end
val intintmap_to_list : 'a IntIntMap.t -> (IntIntMap.key * 'a) list
val intintmap_string_of_t : 'a -> 'b -> string
```

5.6 Hash

55 ⟨*common.mli for collection types* 46⟩+≡

```
(*****************************************************************************)
(* Hash *)
(*****************************************************************************)

(* Note that Hashtbl keep old binding to a key so if want a hash
 * of a list, then can use the Hashtbl as is. Use Hashtbl.find_all then
 * to get the list of bindings
 *
 * Note that Hashtbl module use different convention :( the object is
 * the first argument, not last as for List or Map.
 *)
```

```
(* obsolete: can use directly the Hashtbl module *)
val hcreate : unit -> ('a, 'b) Hashtbl.t
val hadd : 'a * 'b -> ('a, 'b) Hashtbl.t -> unit
val hmem : 'a -> ('a, 'b) Hashtbl.t -> bool
val hfind : 'a -> ('a, 'b) Hashtbl.t -> 'b
val hreplace : 'a * 'b -> ('a, 'b) Hashtbl.t -> unit
val hiter : ('a -> 'b -> unit) -> ('a, 'b) Hashtbl.t -> unit
val hfold : ('a -> 'b -> 'c -> 'c) -> ('a, 'b) Hashtbl.t -> 'c -> 'c
val hremove : 'a -> ('a, 'b) Hashtbl.t -> unit


val hfind_default : 'a -> (unit -> 'b) -> ('a, 'b) Hashtbl.t -> 'b
val hfind_option : 'a -> ('a, 'b) Hashtbl.t -> 'b option
val hupdate_default :
  'a -> ('b -> 'b) -> (unit -> 'b) -> ('a, 'b) Hashtbl.t -> unit

val add1: int -> int
val cst_zero: unit -> int

val hash_to_list : ('a, 'b) Hashtbl.t -> ('a * 'b) list
val hash_to_list_unsorted : ('a, 'b) Hashtbl.t -> ('a * 'b) list
val hash_of_list : ('a * 'b) list -> ('a, 'b) Hashtbl.t


val hkeys : ('a, 'b) Hashtbl.t -> 'a list
```

56 ⟨*common.mli for collection types* 46⟩+≡

```
(*****************************************************************************)
(* Hash sets *)
(*****************************************************************************)

type 'a hashset = ('a, bool) Hashtbl.t


(* common use of hashset, in a hash of hash *)
val hash_hashset_add : 'a -> 'b -> ('a, 'b hashset) Hashtbl.t -> unit

val hashset_to_set :
 < fromlist : ('a ) list -> 'c; .. > -> ('a, 'b) Hashtbl.t -> 'c

val hashset_to_list : 'a hashset -> 'a list
val hashset_of_list : 'a list -> 'a hashset
```

5.7 Stack

57a ⟨*common.mli for collection types* 46⟩+≡
```
(*****************************************************************************)
(* Stack *)
(*****************************************************************************)

type 'a stack = 'a list
val empty_stack : 'a stack
val push : 'a -> 'a stack -> 'a stack
val top : 'a stack -> 'a
val pop : 'a stack -> 'a stack

val top_option: 'a stack -> 'a option

val push2 : 'a -> 'a stack ref -> unit
val pop2: 'a stack ref -> 'a
```

57b ⟨*common.mli for collection types* 46⟩+≡
```
(*****************************************************************************)
(* Stack with undo/redo support *)
(*****************************************************************************)

type 'a undo_stack = 'a list * 'a list
val empty_undo_stack : 'a undo_stack
val push_undo : 'a -> 'a undo_stack -> 'a undo_stack
val top_undo : 'a undo_stack -> 'a
val pop_undo : 'a undo_stack -> 'a undo_stack
val redo_undo: 'a undo_stack -> 'a undo_stack
val undo_pop: 'a undo_stack -> 'a undo_stack

val top_undo_option: 'a undo_stack -> 'a option
```

5.8 Trees

57c ⟨*common.mli for collection types* 46⟩+≡
```
(*****************************************************************************)
(* Binary tree *)
(*****************************************************************************)
type 'a bintree = Leaf of 'a | Branch of ('a bintree * 'a bintree)
```

57d ⟨*common.mli for collection types* 46⟩+≡
```
(*****************************************************************************)
(* N-ary tree *)
(*****************************************************************************)
```

```
(* no empty tree, must have one root at least *)
type 'a tree = Tree of 'a * ('a tree) list

val tree_iter : ('a -> unit) -> 'a tree -> unit
```

58 ⟨*common.mli for collection types* 46⟩+≡

```
(*****************************************************************************)
(* N-ary tree with updatable childrens *)
(*****************************************************************************)

(* no empty tree, must have one root at least *)
type 'a treeref =
  | NodeRef of 'a *   'a treeref list ref

val treeref_node_iter:
  (('a * 'a treeref list ref) -> unit) -> 'a treeref -> unit
val treeref_node_iter_with_parents:
  (('a * 'a treeref list ref) -> ('a list) -> unit) ->
  'a treeref -> unit

val find_treeref:
  (('a * 'a treeref list ref) -> bool) ->
  'a treeref -> 'a treeref

val treeref_children_ref:
  'a treeref -> 'a treeref list ref

val find_treeref_with_parents_some:
 ('a * 'a treeref list ref -> 'a list -> 'c option) ->
 'a treeref -> 'c

val find_multi_treeref_with_parents_some:
 ('a * 'a treeref list ref -> 'a list -> 'c option) ->
 'a treeref -> 'c list


(* Leaf can seem redundant, but sometimes want to directly see if
 * a children is a leaf without looking if the list is empty.
 *)
type ('a, 'b) treeref2 =
  | NodeRef2 of 'a * ('a, 'b) treeref2 list ref
  | LeafRef2 of 'b


val find_treeref2:
  (('a * ('a, 'b) treeref2 list ref) -> bool) ->
  ('a, 'b) treeref2 -> ('a, 'b) treeref2

val treeref_node_iter_with_parents2:
  (('a * ('a, 'b) treeref2 list ref) -> ('a list) -> unit) ->
  ('a, 'b) treeref2 -> unit

val treeref_node_iter2:
  (('a * ('a, 'b) treeref2 list ref) -> unit) -> ('a, 'b) treeref2 -> unit

(*


val treeref_children_ref: ('a, 'b) treeref -> ('a, 'b) treeref list ref

val find_treeref_with_parents_some:
 ('a * ('a, 'b) treeref list ref -> 'a list -> 'c option) ->
 ('a, 'b) treeref -> 'c

val find_multi_treeref_with_parents_some:
 ('a * ('a, 'b) treeref list ref -> 'a list -> 'c option) ->
 ('a, 'b) treeref -> 'c list
*)
```

5.9 Graph

59 ⟨*common.mli for collection types* 46⟩+≡

```
(*****************************************************************************)
(* Graph. But have a look too at Ograph_*.mli; it's better *)
(*****************************************************************************)

type 'a graph = 'a set * ('a * 'a) set

val add_node : 'a -> 'a graph -> 'a graph
val del_node : 'a -> 'a graph -> 'a graph

val add_arc : 'a * 'a -> 'a graph -> 'a graph
val del_arc : 'a * 'a -> 'a graph -> 'a graph

val successors : 'a -> 'a graph -> 'a set
val predecessors : 'a -> 'a graph -> 'a set

val nodes : 'a graph -> 'a set

val fold_upward : ('a -> 'b -> 'a) -> 'b set -> 'a -> 'b graph -> 'a
val empty_graph : 'a list * 'b list
```

5.10 Generic op

60 ⟨*common.mli for collection types* 46⟩+≡

```
(*****************************************************************************)
(* Generic op *)
(*****************************************************************************)

(* mostly alias to functions in List *)

val map : ('a -> 'b) -> 'a list -> 'b list
val filter : ('a -> bool) -> 'a list -> 'a list
val fold : ('a -> 'b -> 'a) -> 'a -> 'b list -> 'a

val member : 'a -> 'a list -> bool

val iter : ('a -> unit) -> 'a list -> unit

val find : ('a -> bool) -> 'a list -> 'a

val exists : ('a -> bool) -> 'a list -> bool
val forall : ('a -> bool) -> 'a list -> bool

val big_union : ('a -> 'b set) -> 'a list -> 'b set

(* same than [] but easier to search for, because [] can also be a pattern *)
val empty_list : 'a list

val sort : ('a -> 'a -> int) -> 'a list -> 'a list

val length : 'a list -> int

val null : 'a list -> bool

val head : 'a list -> 'a
val tail : 'a list -> 'a list

val is_singleton : 'a list -> bool
```

Chapter 6

Misc

61a ⟨*common.mli misc* 61a⟩≡

```
(*xxxx*)

⟨common.mli misc other 65b⟩
```

6.1 Geometry

61b ⟨*common.mli misc* 61a⟩+≡

```
(****************************************************************************)
(* Geometry (ICFP raytracer) *)
(****************************************************************************)

type vector = float * float * float

type point = vector
type color = vector

val dotproduct : vector * vector -> float

val vector_length : vector -> float

val minus_point : point * point -> vector

val distance : point * point -> float

val normalise : vector -> vector

val mult_coeff : vector -> float -> vector
val add_vector : vector -> vector -> vector
val mult_vector : vector -> vector -> vector
val sum_vector : vector list -> vector
```

6.2 Pics

62a ⟨*common.mli misc* 61a⟩+≡

```
(*****************************************************************************)
(* Pics (ICFP raytracer) *)
(*****************************************************************************)
type pixel = int * int * int
val write_ppm : int -> int -> pixel list -> filename -> unit
val test_ppm1 : unit -> unit
```

6.3 Diff

62b ⟨*common.mli misc* 61a⟩+≡

```
(*****************************************************************************)
(* Diff (LFS) *)
(*****************************************************************************)

type diff = Match | BnotinA | AnotinB
val diff : (int -> int -> diff -> unit) -> string list * string list -> unit
val diff2 : (int -> int -> diff -> unit) -> string * string -> unit
```

6.4 Parsers

62c ⟨*common.mli misc* 61a⟩+≡

```
(*****************************************************************************)
(* Parsers (aop-colcombet)                                                 *)
(*****************************************************************************)

val parserCommon : Lexing.lexbuf -> ('a -> Lexing.lexbuf -> 'b) -> 'a -> 'b
val getDoubleParser :
  ('a -> Lexing.lexbuf -> 'b) -> 'a -> (string -> 'b) * (string -> 'b)
```

62d ⟨*common.mli misc* 61a⟩+≡

```
(*****************************************************************************)
(* Parsers (cocci) *)
(*****************************************************************************)
```

```
(* Currently lexing.ml does not handle the line number position.
 * Even if there is some fields in the lexing structure, they are not
 * maintained by the lexing engine :( So the following code does not work:
 *
 *   let pos = Lexing.lexeme_end_p lexbuf in
 *   sprintf "at file %s, line %d, char %d" pos.pos_fname pos.pos_lnum
 *      (pos.pos_cnum - pos.pos_bol) in
 *
 * Hence those functions to overcome the previous limitation.
 *)

type parse_info = {
    str: string;
    charpos: int;

    line: int;
    column: int;
    file: filename;
  }
val fake_parse_info : parse_info
val string_of_parse_info : parse_info -> string
val string_of_parse_info_bis : parse_info -> string

(* array[i] will contain the (line x col) of the i char position *)
val full_charpos_to_pos : filename -> (int * int) array

(* fill in the line and column field of parse_info that were not set
 * during lexing because of limitations of ocamllex. *)
val complete_parse_info :
  filename -> (int * int) array -> parse_info -> parse_info

val full_charpos_to_pos_large:
  filename -> (int -> (int * int))

val complete_parse_info_large :
  filename -> (int -> (int * int))  -> parse_info -> parse_info

(* return line x col x str_line  from a charpos. This function is quite
 * expensive so don't use it to get the line x col from every token in
 * a file. Instead use full_charpos_to_pos.
 *)
val info_from_charpos : int -> filename -> (int * int * string)

val error_message :       filename -> (string * int) -> string
val error_message_short : filename -> (string * int) -> string
val error_message_info :  parse_info -> string
```

```
(* add a 'decalage/shift' argument to handle stuff such as cpp which includes
 * files and who can make shift.
 *)
val error_messagebis : filename -> (string * int) -> int -> string
```

6.5 Scope

64 ⟨*common.mli misc* 61a⟩+≡

```
(*****************************************************************************)
(* Scope managment (cocci) *)
(*****************************************************************************)

(* for example of use, see the code used in coccinelle *)
type ('a, 'b) scoped_env = ('a, 'b) assoc list

val lookup_env : (* Eq a *) 'a -> ('a, 'b) scoped_env -> 'b
val member_env_key : 'a -> ('a, 'b) scoped_env -> bool

val new_scope : ('a, 'b) scoped_env ref -> unit
val del_scope : ('a, 'b) scoped_env ref -> unit

val do_in_new_scope : ('a, 'b) scoped_env ref -> (unit -> unit) -> unit

val add_in_scope : ('a, 'b) scoped_env ref -> 'a * 'b -> unit




(* for example of use, see the code used in coccinelle *)
type ('a, 'b) scoped_h_env = {
  scoped_h : ('a, 'b) Hashtbl.t;
  scoped_list : ('a, 'b) assoc list;
}
val empty_scoped_h_env : unit -> ('a, 'b) scoped_h_env
val clone_scoped_h_env : ('a, 'b) scoped_h_env -> ('a, 'b) scoped_h_env

val lookup_h_env : 'a -> ('a, 'b) scoped_h_env -> 'b
val member_h_env_key : 'a -> ('a, 'b) scoped_h_env -> bool

val new_scope_h : ('a, 'b) scoped_h_env ref -> unit
val del_scope_h : ('a, 'b) scoped_h_env ref -> unit

val do_in_new_scope_h : ('a, 'b) scoped_h_env ref -> (unit -> unit) -> unit
val add_in_scope_h : ('a, 'b) scoped_h_env ref -> 'a * 'b -> unit
```

6.6 Terminal

65a ⟨*common.mli misc* 61a⟩+≡

```
(*****************************************************************************)
(* Terminal (LFS) *)
(*****************************************************************************)

(* don't forget to call Common_extra.set_link () *)

val _execute_and_show_progress_func :
  (int (* length *) -> ((unit -> unit) -> unit) -> unit) ref
val execute_and_show_progress :
 int (* length *) -> ((unit -> unit) -> unit) -> unit
```

6.7 Other, Db, GUI, Graphics

65b ⟨*common.mli misc other* 65b⟩≡

```
(*****************************************************************************)
(* DB (LFS) *)
(*****************************************************************************)

(* cf oassocbdb.ml or oassocdbm.ml  *)

(*****************************************************************************)
(* GUI (LFS, CComment, otimetracker) *)
(*****************************************************************************)

(* cf ocamlgtk and my gui.ml *)


(*****************************************************************************)
(* Graphics (otimetracker) *)
(*****************************************************************************)

(* cf ocamlgl and my opengl.ml *)
```

Part II

OCommon

Chapter 7

Overview

[1]

[2]

67 ⟨*objet.mli* 67⟩≡

```
(* TypeClass via objects. Cf also now interfaces.ml *)
class virtual objet :
object('o)
  method invariant: unit -> unit
  (* method check: unit -> unit *)

  method of_string: string -> unit
  method to_string: unit -> string
  method debug: unit -> unit

  (* ugly (but convenient): those methods allow to extend an interface without
   * changing its interface. For instance in oassocbtree I want to
   * provide a method to commit, but doing so will mean break the interface
   * of oassoc. But if provide the commit code via a misc_op_hook, then
   * I will not break the interface.
   *)
  method misc_op_hook: unit -> 'o
  method misc_op_hook2: unit
end
```

[1]TODO: SEMI objet.ml cf also interface.ml ofullcommon.ml

[2]TODO: ocollection.ml SEMI ocollection.mli oarray.ml SEMI oarray.mli oassoc.ml SEMI oassoc.mli osequence.ml SEMI osequence.mli oset.ml SEMI oset.mli ograph.ml SEMI ograph.mli ograph_extended.ml SEMI ograph_extended.mli ograph_simple.ml SEMI ograph_simple.mli
seti.ml

Chapter 8

Ocollection

```
(*
 * The derived classes of collections:
 * - sequence(next, nth): array, list, stack, queue, and mixed
 *   (fast cons, fast snoc, fast append, cf okasaki)
 * - set(union): setl, setb, seti, seth
 * - assoc(find): assocl, mapb, hash, btree, multimap (mais bof, can do
 *   with map of set)
 * - graph: graph1way, graph2way, graphref, graphmatrix?
 *
 * Some features/notes:
 * - views a la wadler to make it cool (I hate get/set).
 * - take list in parameters to be able to construct value as is easily
 * - take the comparaison function in parameters (=> functorial set made cool)
 *   make l [], h [], ... as in perl, and pass the func from pervasive
 *   in oo form (list, ...)
 * - pure/impure: could put 2 interface, with one that show that inpure
 *   by making the operation return unit, but simpler to have one interface.
 * - the core method and default method (via virtual classes)
 *   better to use virtual than typeclass, virtual play both roles:
 *   an interface and default code
 *
 * - pb binary methods: use tosetb tricks, or via (not safe) Obj.magic.
 * - array/list are both a sequence _and_ a dictionnary, so are both
 *   a collection(a) and a collection(i,a) at the same time. But cant do that.
 *   So for array, I see it mainly as an assoc => favor assoc, and
 *   for list, I see it mainly as a collection => favor collection
 *
 * ??mixins: comparable, iterator, via virtual class in ocaml
 * ?? kind of haskell class + default value
 *
 * ?? persistence, caching, peut prendre en param le type de map qu'il cache,
 * comme en perl, evite du marshalling kan wrapped = bdb.
 *
 * ?? lazy wrapper,  how avoid complexity of having to define each time
 * a hashP, hashC, hashL, hashPCL, ... ?
 *
 * ?? I define those classes cos their name are cool, say what is intended to
 * do with
 *
 * todo: cf book on algo, a la rivest/sedgewick
 * todo: recreate collection hierarchy, inspire smalltalk ? haskell ? merd ?
 * todo: put a clean sequence (inherit collection) and make array a special
 * class
 * todo: make ostack (FIFO), oqueue (LIFO)
 *
 *
 * influences: okasaki, merd (pixel), java classes, smalltalk classes
 *)
```

69 ⟨*ocollection.mli* 69⟩≡

```
type ('a, 'b) view =
  | Empty
  | Cons of 'a * 'b

class virtual ['a] ocollection :
object ('o)
  inherit Objet.objet

  method virtual empty : 'o
  method virtual add : 'a -> 'o

  method virtual iter : ('a -> unit) -> unit
  method virtual view : ('a, 'o) view

  (* no need virtual, but better to force redefine for efficiency *)
  method virtual del : 'a -> 'o
  method virtual mem : 'a -> bool
  method virtual null : bool

  (* effect version *)
  method add2: 'a -> unit
  method del2: 'a -> unit
  method clear: unit


  method fold : ('c -> 'a -> 'c) -> 'c -> 'c
  method fromlist : 'a list -> 'o
  method tolist : 'a list

  method exists : ('a -> bool) -> bool
  method filter : ('a -> bool) -> 'o

  method length : int

  method getone : 'a
  method others : 'o
end
```

Chapter 9

Oset

[1]

71 ⟨*oset.mli* 71⟩≡
```
class virtual ['a] oset :
object ('o)
  inherit ['a] Ocollection.ocollection

  method cardinal : int

  method virtual inter : 'o -> 'o
  method virtual minus : 'o -> 'o
  method virtual union : 'o -> 'o

  method is_singleton : bool
  method is_subset_of : 'o -> bool
  method is_equal : 'o -> bool

  method virtual toset : 'd
  method tosetb : 'a Setb.t
  method toseti : Seti.seti
  method tosetpt : SetPt.t
end

val ( $??$ )  : 'a -> < mem : 'a -> bool; .. > -> bool
val ( $++$ )  : < union : 'a -> 'o; .. > -> 'a -> 'o
val ( $**$ )  : < inter : 'a -> 'o; .. > -> 'a -> 'o
val ( $--$ )  : < minus : 'a -> 'o; .. > -> 'a -> 'o
val ( $<<=$ ) : < is_subset_of : 'a -> bool; .. > -> 'a -> bool
val ( $==$ )  : < is_equal : 'a -> bool; .. > -> 'a -> bool
```

[1]TODO: `setxxx`

```
val mapo : ('a -> 'o) -> 'o oset -> 'a oset -> 'o oset
```

Chapter 10

Oassoc

73 ⟨*oassoc.mli* 73⟩≡

```
class virtual ['a, 'b] oassoc :
object ('o)
  inherit ['a * 'b] Ocollection.ocollection

  method virtual assoc : 'a -> 'b
  method virtual delkey : 'a -> 'o

  (* may raise NotFound *)
  method find : 'a -> 'b
  method find_opt: 'a -> 'b option

  method haskey : 'a -> bool
  method replkey : 'a * 'b -> 'o

  (* better to implement it yourself *)
  method virtual keys: 'a list

  method apply : 'a -> ('b -> 'b) -> 'o
  method apply_with_default : 'a -> ('b -> 'b) -> (unit -> 'b) -> 'o

  (* effect version *)
  method apply_with_default2 : 'a -> ('b -> 'b) -> (unit -> 'b) -> unit

end
```

Chapter 11

Osequence

74 ⟨*osequence.mli* 74⟩≡

```
class virtual ['a] osequence :
object ('o)
  inherit [int, 'a] Oassoc.oassoc

  method virtual nth : int -> 'a
  method virtual first : 'a
  method virtual last : 'a
end
```

Chapter 12

Oarray

75 ⟨*oarray.mli* 75⟩≡
```
(* !!take care!!, this is not a pure data structure *)

class ['a] oarray : int -> 'a ->
object ('o)
  inherit ['a] Osequence.osequence

  (* ocollection concrete instantiation of virtual methods *)
  method empty : 'o
  method add : (int * 'a) -> 'o

  method iter : (int * 'a -> unit) -> unit
  method view : (int * 'a, 'o) Ocollection.view

  method del : (int * 'a) -> 'o
  method mem : int * 'a -> bool
  method null : bool


  (* oassoc concrete instantiation of virtual methods *)
  method assoc : int -> 'a
  method delkey : int -> 'o

  method keys: int list

  (* osequence concrete instantiation of virtual methods *)
  method first : 'a
  method last : 'a
  method nth : int -> 'a

end
```

Chapter 13

Ograph

77a ⟨*ograph.mli* 77a⟩≡

```
class virtual ['a] ograph :
object ('o)
  method virtual empty : 'o

  method virtual add_node : 'a -> 'o
  method virtual del_node : 'a -> 'o

  method virtual add_arc : 'a * 'a -> 'o
  method virtual del_arc : 'a * 'a -> 'o


  method virtual nodes : 'a Oset.oset
  method virtual predecessors : 'a -> 'a Oset.oset
  method virtual successors : 'a -> 'a Oset.oset

  method virtual ancestors : 'a Oset.oset -> 'a Oset.oset
  method virtual brothers : 'a -> 'a Oset.oset
  method virtual children : 'a Oset.oset -> 'a Oset.oset

  method mydebug : ('a * 'a list) list
end
```

77b ⟨*ograph_simple.mli* 77b⟩≡

```
open Common

(* essentially a convenient way to access a hash and its reverse hash *)

class ['key, 'node, 'edge] ograph_mutable :
object ('o)
```

```
    method add_node : 'key -> 'node -> unit
    method del_node : 'key -> unit
    method replace_node: 'key -> 'node -> unit
    method add_node_if_not_present: 'key -> 'node -> unit


    method add_arc : ('key * 'key) -> 'edge -> unit
    method del_arc : ('key * 'key) -> 'edge -> unit

    method nodes : ('key, 'node) Oassoc.oassoc

    method successors : 'key -> ('key * 'edge) Oset.oset
    method predecessors : 'key -> ('key * 'edge) Oset.oset
    method allsuccessors : ('key, ('key * 'edge) Oset.oset) Oassoc.oassoc


    method del_leaf_node_and_its_edges: 'key -> unit
    method ancestors : 'key -> 'key Oset.oset
    method leaf_nodes : unit -> 'key Oset.oset

  end

  val print_ograph_generic:
    str_of_key:('key -> string) ->
    str_of_node:('key -> 'node -> string) ->
    Common.filename ->
    ('key, 'node,'edge) ograph_mutable ->
    unit


* graph structure:
*  -  node: index -> nodevalue
*  -  arc: (index * index) * edgevalue
*
* invariant: key in pred is also in succ (completness) and value in
* either assoc is a key also.
*
* How ? matrix ? but no growing array :(
*
* When need index ? Must have an index when can't just use nodevalue
* as a key, cos sometimes may have 2 times the same key, but it must
* be 2 different nodes. For instance in program f(); f(); we want 2
* nodes, one per f(); hence the index. If each node is different,
* then no problem, can omit index.
*
```

79 ⟨*ograph_extended.mli* 79⟩≡
```
open Common

type nodei = int

(* graph structure:
 *  - node: index -> nodevalue
 *  - arc: (index * index) * edgevalue
 *
 * How ? matrix ? but no growing array :(
 *
 * When need index ? Must have an index when can't just use the nodevalue
 * as a key, cos sometimes may have 2 times the same key, but it must
 * be 2 different nodes. For instance in a C program 'f(); f();' we want 2
 * nodes, one per 'f();' hence the index. If each node is different, then
 * no problem, can omit index.
 *)

class ['node, 'edge] ograph_extended :
object ('o)
  method add_node : 'node -> 'o * nodei
  method add_nodei : nodei -> 'node -> 'o * nodei
  method replace_node : nodei * 'node -> 'o
  method del_node : nodei -> 'o

  method add_arc : (nodei * nodei) * 'edge -> 'o
  method del_arc : (nodei * nodei) * 'edge -> 'o

  method nodes : (nodei, 'node) Oassoc.oassoc

  method successors : nodei -> (nodei * 'edge) Oset.oset
  method predecessors : nodei -> (nodei * 'edge) Oset.oset
  method allsuccessors : (nodei, (nodei * 'edge) Oset.oset) Oassoc.oassoc
end


class ['node, 'edge] ograph_mutable :
object ('o)
  method add_node : 'node -> nodei
  method add_nodei : nodei -> 'node -> unit
  method replace_node : nodei * 'node -> unit
  method del_node : nodei -> unit

  method add_arc : (nodei * nodei) * 'edge -> unit
  method del_arc : (nodei * nodei) * 'edge -> unit
  method nodes : (nodei, 'node) Oassoc.oassoc

  method successors : nodei -> (nodei * 'edge) Oset.oset
  method predecessors : nodei -> (nodei * 'edge) Oset.oset
  method allsuccessors : (nodei, (nodei * 'edge) Oset.oset) Oassoc.oassoc
end


val dfs_iter :
  nodei -> (nodei -> unit) -> ('node, 'edge) ograph_mutable -> unit

val dfs_iter_with_path :
  nodei -> (nodei -> nodei list -> unit) -> ('node, 'edge) ograph_mutable ->
  unit

val print_ograph_mutable_generic :
  ('node, 'edge) ograph_mutable ->
  string option -> (* label for the entire graph *)
  (* what string to print for a node and how to color it *)
  ((nodei * 'node) -> (string * string option * string option)) ->
  output_file:filename ->
  launch_gv:bool ->
  unit


val print_ograph_extended :
  ('node * string, 'edge) ograph_extended ->
  filename (* output file *) ->
  bool (* launch gv ? *) ->
  unit

val print_ograph_mutable :
  ('node * string, 'edge) ograph_mutable ->
  filename (* output file *) ->
  bool (* launch gv ? *) ->
  unit

val launch_gv_cmd : Common.filename -> unit
```

Chapter 14

Odb

Part III

Extra Common

[1] [2]

[3]

[1]TODO: `macro.ml4`

[2]TODO: `common_extra.ml`

[3]TODO: `concurrency.ml distribution.ml graphic.ml gui.ml opengl.ml`

Chapter 15

Interface

[1]

84 ⟨*interfaces.ml* 84⟩≡
```
open Common.BasicType

(*****************************************************************************)
(* Type classes via module signature. *)
(*****************************************************************************)
(*
 * Use this not so much for functors, I hate functors, but
 * more to force me to have consistent naming of stuff.
 *
 * It's related to objet.ml in some way, but use a different scheme.
 *
 * src: (strongly) inspired by Jane Street core lib, which in turn
 * may have been strongly inspired by Java Interfaces or Haskell
 * type classes.
 *
 *
 *
 * Example of use in .mli:
 *
 *   open Interfaces
 *   include Stringable with type stringable = t
 *   include Comparable with type comparable = t
 *
 * Example of use in .ml:
 *
 *   type xxx
 *   type stringable = xxx
```

[1] TODO: `(cf also objet.ml)`

```
 *   let of_string = bool_of_string
 *   let to_string = string_of_bool
 *
 *
 * No this file is not about (graphical) user interface. See gui.ml for that.
 *
 *
 * todo? but as in type class, or object, can not have default method
 * with this scheme ?
 *)



(*****************************************************************************)
(* Basic *)
(*****************************************************************************)

(* note: less need for cloneable, copyable as in Java. Only needed
 * when use ref, but refs should be avoided anyway so better not to
 * encourage it.
 *
 * Often found this in haskell:
 *
 *    data x = ... deriving (Read, Show, Eq, Ord, Enum, Bounded)
 *
 * Apparently this is what is considered basic by haskell.
 *)


module type Check_able = sig
  type checkable
  val invariant: checkable -> unit (* raise exception *)
end



(* Normally should not use the '=' of ocaml. cf common.mli on this issue. *)
module type Eq_able = sig
  type eqable
  val equal : eqable -> eqable -> bool
  (* Jane Street have far more (complex) stuff for this typeclass *)

  val (=*=): eqable -> eqable -> bool
end
```

```
(* Same, should not use compare normally, dangerous when evolve code.
 * Called Ord in haskell. Inherit Eq normally.
 *)
module type Compare_able = sig
  type compareable
  val compare: compareable -> compareable -> bool
end
(* Jane street have also some binable, sexpable *)


(* Haskell have lots of related type class after Num such as
 * Real, Fractional, Integral, RealFrac, Floating, RealFloat
 *)
module type Num_able = sig
  type numable
  (* +, -, etc *)
end



(*****************************************************************************)
(* Show/read related *)
(*****************************************************************************)


(* Called show/read in haskell *)
module type String_able = sig
  type stringable
  val of_string : string -> stringable
  val to_string : stringable -> string
end

module type Debug_able = sig
    type debugable
    val debug: debugable -> string
end


module type XML_able = sig
    type xmlable
    val of_xml: string -> xmlable
    val to_xml: xmlable -> string
end
(* Jane street have also some BIN_able, and SEXP_able (but no sex_able) *)
```

```
module type File_able = sig
    type fileable
    val load: filename -> fileable
    val save: fileable -> filename -> unit
end

(* a.k.a Marshall_able *)
module type Serialize_able = sig
    type serializeable
    val serialize: serializeable -> string
    val unserialize: string -> serializeable
end


module type Open_able = sig
    type openable
    val openfile: filename -> openable
    val close: openable -> unit
end

(*****************************************************************************)
(* Other *)
(*****************************************************************************)

(* This is related to ocollection.ml in some way, but use a different scheme *)

(* Require Constructor class ? So can not do it ? apparently can. Note the
 * 'b which is not declareted but seems to pose no problem to ocamlc.
 *)
module type Map_able = sig
    type 'a mapable
    val map: ('a -> 'b) -> 'a mapable -> 'b mapable
end

module type Iter_able = sig
    type 'a iterable
    val iter: ('a -> unit) -> 'a iterable -> unit
end


(* testable ? actionable ? *)

(* *)

(* monad ? functor *)
```

```
(*****************************************************************************)
(* Idea taken from Jane Street Core library, slightly changed.
 *
 * It's another way to organize data structures, module instead of objects.
 * It's also the Java way.
 *
 * It makes some code looks a little bit like Haskell* typeclass.
 *
 *)

(* In Jane Street they put each interface in its own file but then have to
 * do that:
 *
 * module type Stringable = Stringable.S
 * module type Comparable = Comparable.S
 * module type Floatable = Floatable.S
 * module type Hashable = Hashable.S
 * module type Infix_comparators = Comparable.Infix
 * module type Monad = Monad.S
 * module type Robustly_comparable = Robustly_comparable.S
 * module type Setable = Setable.S
 * module type Sexpable = Sexpable.S
 * module type Binable = Binable.S
 *
 * And I dont like having too much files, especially as all those xxable
 * end with able, not start, so don't see them together in the directory.
 *)
```

Chapter 16

Concurrency

89 ⟨*concurrency.ml* 89⟩≡
```
open Common


let _biglock = Mutex.create ()
let atomic f =
  Mutex.lock _biglock;
  Common.finalize f (fun () -> Mutex.unlock _biglock)
```

Chapter 17

Distribution

Chapter 18

Graphic

```
(* alternatives:
 * - Graphics module of ocaml, very good for prototyping (used in
 *   icfp contest)
 * - opengl
 * - gtk drawing area
 * - sdl ?
 *)
```

Chapter 19

OpenGL

Chapter 20

GUI

```
GUI via lablgtk.
 *
 * Alternatives:
 *  - tk, but tk ... a little bit old style
 *  - qt, but poor wrapper
 *  - wxwindow, but poor wrapper or inexistant
 * => lablgtk seems the most mature.
 *
 * cf also ocaml.org library notes on lablgtk.
 *
 *
```

Chapter 21

ParserCombinators

parser_combinators.ml

```
(* src: Jon Harrop.
 *
 * "Certain applications are extremely well suited to functional
 * programming and parsing is one of them. Specifically, the ability to
 * write functional combinators that allow parsers for everything from
 * integers up to symbolic expressions to be composed is more general
 * and provides more opportunity for code reuse than the use of
 * conventional parser generators such as ocamllex and ocamlyacc. This
 * article explains how parser combinators may be designed and
 * implemented in OCaml, using the standard example of a calculator."
 *
 * Based on haskell articles I guess like meijer functional pearl or
 * graham hutton articles. Also maybe based on haskell parsec.
 *
 * pad: a few bugfix. I also put more restrictive and descriptive types.
 * pad: I remember having coded such a library, maybe not in ocaml.
 * Or maybe it was during a "TP compilation" at INSA ? I remember having
 * a generic lexer. Or maybe it was genlex ?
 *
 *
 *
 *
 * alternatives: genlex + parser extension of ocaml (streams).
 * cf genlex doc:
 *
 * Example: a lexer suitable for a desk calculator is obtained by
 * let lexer = make_lexer ["+";"-";"*";"/";"let";"="; "("; ")"]
 * let parse_expr = parser
 *     [< 'Int n >] -> n
 *   | [< 'Kwd "("; n = parse_expr; 'Kwd ")" >] -> n
 *   | [< n1 = parse_expr; n2 = parse_remainder n1 >] -> n2
 * and parse_remainder n1 = parser
 *   [< 'Kwd "+"; n2 = parse_expr >] -> n1+n2
 *   | ...
 * type token =
 * |    Kwd of string
 * |    Ident of string
 * |    Int of int
 * |    Float of float
 * |    String of string
 * |    Char of char
 *
 *
 * Cf also ocaml manual
 *  let rec parse_expr = parser
 *      [< e1 = parse_mult; e = parse_more_adds e1 >] -> e
 *  and parse_more_adds e1 = parser
 *      [< 'Kwd "+"; e2 = parse_mult; e = parse_more_adds (Sum(e1, e2)) >] -> e
 *    | [< 'Kwd "-"; e2 = parse_mult; e = parse_more_adds (Diff(e1, e2)) >] -> e
 *    | [< >] -> e1
 *  and parse_mult = parser
 *      [< e1 = parse_simple; e = parse_more_mults e1 >] -> e
 *  and parse_more_mults e1 = parser
 *      [< 'Kwd "*"; e2 = parse_simple; e = parse_more_mults (Prod(e1, e2)) >] -> e
 *    | [< 'Kwd "/"; e2 = parse_simple; e = parse_more_mults (Quot(e1, e2)) >] -> e
 *    | [< >] -> e1
 *  and parse_simple = parser
 *      [< 'Ident s >] -> Var s
 *    | [< 'Int i >] -> Const(float i)
 *    | [< 'Float f >] -> Const f
 *    | [< 'Kwd "("; e = parse_expr; 'Kwd ")" >] -> e;;
 * But see how they are forced to use a LL(1) grammar which denatures the
 * grammar "parse_more_xxx"
 *
```

95 ⟨*parser_combinators.mli* 95⟩≡

```
(*****************************************************************************)
(* src: Jon Harrop.
 *
 * "Certain applications are extremely well suited to functional
 * programming and parsing is one of them. Specifically, the ability to
 * write functional combinators that allow parsers for everything from
 * integers up to symbolic expressions to be composed is more general
 * and provides more opportunity for code reuse than the use of
 * conventional parser generators such as ocamllex and ocamlyacc. This
 * article explains how parser combinators may be designed and
 * implemented in OCaml, using the standard example of a calculator."
 *
 * pad: a few bugfixes. I also put more restrictive and descriptive types.
 *
 *)

(*****************************************************************************)

(* A generic parser takes a list of stuff (either char for lexical
 * parser or tokens for grammar parser) and return something and the
 * remaing list of stuff. *)
type ('a, 'b) genp = 'a list -> 'b * 'a list
val val_of_parser : 'b * 'a list -> 'b

(* lexer = parser of char list *)
(* type 'a lexer = (char, 'a) genp *)

(* grammer = parser ot tokens *)
(* type 'a pparser = (token, 'a) genp *)


val ( ||| ) : ('a, 'b) genp -> ('a, 'b) genp -> ('a, 'b) genp
(* ('a -> 'b) -> ('a -> 'b) -> 'a -> 'b *)
val ( +++ ) : ('a, 'b) genp -> ('a, 'c) genp -> ('a,   'b * 'c) genp
(* ('a -> 'b * 'c) -> ('c -> 'd * 'e) -> 'a -> ('b * 'd) * 'e *)

val many : ('a, 'b) genp -> ('a, 'b list) genp
(* ('a -> 'b * 'a) -> 'a -> 'b list * 'a *)

val ( >| ) : ('a, 'b) genp -> ('b -> 'c) -> ('a, 'c) genp
(* ('a -> 'b * 'c) -> ('b -> 'd) -> 'a -> 'd * 'c *)

(* was called 'some', but confusing *)
val pred : ('a -> bool) -> ('a, 'a) genp
(* ('a -> bool) -> 'a list -> 'a * 'a list *)

val a : 'a -> ('a, 'a) genp
(* 'a -> 'a list -> 'a * 'a list *)

val several : ('a -> bool) -> ('a, 'a list) genp
(* ('a -> bool) -> 'a list -> 'a list * 'a list *)


module Abstr : sig
    type t
    val x : t
end

val fin : ('a, Abstr.t) genp
(* 'a list -> Abstr.t * 'b list *)


val digit    : char -> bool
val alpha    : char -> bool
val symbol   : char -> bool
val alphanum : char -> bool
val space    : char -> bool

val alphanum_underscore : char -> bool
val alphanum_minus : char -> bool
val alphanum_under_minus : char -> bool

val collect : char * char list -> string
val list_of_string : string -> char list


(******************************************************************************)
type token =
  | IDENT of string
  | KWD of string
  | INT of string
  | SYM of string
  | STR of string

val string_of_token : token -> string

type lexer = (char, token) genp

val rawident : lexer
(* char list -> token * char list *)
val rawnumber : lexer
(* char list -> token * char list *)

val rawsymbol : lexer

(* not space, not digit *)
val rawkeyword : lexer
(* char list -> token * char list *)

val rawstring : lexer
val lex_gen : lexer -> string -> token list

(******************************************************************************)
val token : lexer
(* char list -> token * char list *)
val tokens : (char, token list) genp
(* char list -> token list * char list *)

val alltokens : (char, token list) genp
(* char list -> token list * 'a list *)

val lex : string -> token list


(******************************************************************************)
(* cant use parser as it's a reseverd word *)
type 'a pparser = (token, 'a) genp

val ident : string pparser
(* token list -> string * token list *)
val int :  string pparser
(* token list -> string * token list *)
val string : string pparser

type expr =
    | Int of int
    | Var of string
    | Add of expr * expr
    | Mul of expr * expr

val atom : expr pparser
(* token list -> expr * token list *)
val factor : expr pparser
(* token list -> expr * token list *)
val term : expr pparser
(* token list -> expr * token list *)
val expr : expr pparser
(* token list -> expr * 'a list *)

val parse : 'a pparser -> string -> 'a
(* (token list -> 'a * 'b) -> string -> 'a *)


(******************************************************************************)

module Infix : sig
```

```
  val ( ||| ) : ('a, 'b) genp -> ('a, 'b) genp -> ('a, 'b) genp
    (* ('a -> 'b) -> ('a -> 'b) -> 'a -> 'b *)
  val ( +++ ) : ('a, 'b) genp -> ('a, 'c) genp -> ('a,   'b * 'c) genp
    (* ('a -> 'b * 'c) -> ('c -> 'd * 'e) -> 'a -> ('b * 'd) * 'e *)
  val ( >| ) : ('a, 'b) genp -> ('b -> 'c) -> ('a, 'c) genp
    (* ('a -> 'b * 'c) -> ('b -> 'd) -> 'a -> 'd * 'c *)
end
```

Chapter 22

Backtrace

[1]

```
This function is especially useful with lablgtk which intercepts
 * the exception and forbid them to reach the toplevel, or with LFS
 * where I can not allow any exception to stop mount.lfs.
 *
```

[1] TODO: `backtrace.ml` `backtrace_c.c`

Chapter 23

Glimpse

[1]

101 ⟨*glimpse.mli* 101⟩≡
```
open Common

(* ---------------------------------------------------------------------- *)
type glimpse_search =
  | GlimpseCaseInsensitive
  | GlimpseWholeWord
val default_glimpse_search : glimpse_search list

type glimpsedir = Common.dirname

(* ---------------------------------------------------------------------- *)
val glimpseindex : string -> dirname -> glimpsedir -> unit

val glimpseindex_files : filename list -> glimpsedir -> unit


(* ---------------------------------------------------------------------- *)
val glimpse :
  string -> ?options:glimpse_search list -> glimpsedir -> filename list
val grep : 'a -> 'b


(* ---------------------------------------------------------------------- *)
val check_have_glimpse : unit -> unit

val s_of_glimpse_search : glimpse_search -> string
val s_of_glimpse_options : glimpse_search list -> string
```

[1]TODO: `glimpse.ml`

Chapter 24

Regexp

[1]

```
(* because Str of ocaml sux with word boundary, newline, etc.
 * ex: \\bcia_pci_tbi_try2\\b
 *)
```

[1]TODO: `regexp.ml`

Chapter 25

Sexp and binio

[1]

[1]TODO: `bin_common.ml sexp_common.ml`

Chapter 26

Python

[1]

```
 * Cf also ocaml.org library notes on pycaml.
 *
 * As there are lots of libraries or library wrappers written in python,
 * can conveniently get access to them, for instance nltk, with
 * this module. Easier to talk to a lib through a python binding than
 * to talk to the lib through the C foreign mechanism of Ocaml.
 * But must still install the python dependencies on the client.
 * Can not just distribute the ocaml binaries and ldd librairies
 * dependencies. There is unfortunately hidden dependencies
 * with the use of python.ml. So tradeoff.
 *
```

[1]TODO: `python.ml python_ocaml.py`

Conclusion

Appendix A

Indexes

Appendix B
References

Bibliography

[1] George Necula, *CIL*, CC. `http://manju.cs.berkeley.edu/cil/`