Signature des fonctions codées

Fichier *array.ml*

```
module type VECT =
  sig
    type 'a vect
    val vect_of_array : 'a array -> 'a vect
    val array_of_vect : 'a vect -> 'a array
    val length : 'a vect -> int
    val create : 'a -> int -> 'a vect
    val create_empty : 'a -> 'a vect
    val make_matrix : 'a -> int -> int -> 'a vect vect
    val alternate_id : int -> 'a -> ('a -> 'a) -> 'a vect
    val extend : 'a vect -> int -> 'a -> 'a vect
    val affect : 'a vect -> 'a -> int -> unit
    val value : 'a vect -> int -> 'a
    val reverse : 'a vect -> 'a vect
    val concat : 'a vect -> 'a vect -> 'a vect
    val sum : 'a vect -> 'a vect -> ('a -> 'a -> 'a) -> 'a vect
    val prod : 'a vect -> 'a vect -> ('a -> 'a -> 'a) -> 'a vect
    val sub_vect : 'a vect -> int -> int -> 'a vect
    val put_in : 'a vect -> 'a vect -> int -> unit
    val dilate : 'a vect -> int -> 'a -> 'a vect
    val extr_vect : 'a vect -> int -> int -> int -> 'a vect
    val filter : 'a vect -> 'a vect -> 'a vect ->
      ('a -> 'a -> 'a) -> ('a -> 'a -> 'a) -> ('a -> 'a) -> 'a -> 'a vect
  end

module type MATRIX =
  sig
    type 'a matrix
    val dim : 'a matrix -> int * int
    val create : 'a -> int -> int -> 'a matrix
    val matrix_of_vect : 'a Vect.vect -> 'a matrix
    val value : 'a matrix -> int * int -> 'a
    val affect : 'a matrix -> 'a -> int * int -> unit
    val vect : 'a matrix -> int -> 'a Vect.vect
    val affect_vect : 'a matrix -> 'a Vect.vect -> int * int -> unit
    val put_in : 'a matrix -> 'a matrix -> int * int -> unit
    val line : 'a matrix -> int -> 'a Vect.vect
    val affect_line : 'a matrix -> 'a Vect.vect -> int * int -> unit
    val sum : 'a matrix -> 'a matrix -> ('a -> 'a -> 'a) -> 'a matrix
    val prod_scal_cano : 'a matrix -> 'a matrix -> ('a -> 'a -> 'a) ->
      ('a -> 'a -> 'a) -> 'a -> 'a
    val norm_eucli : 'a matrix -> ('a -> 'a -> 'a) -> ('a -> 'a -> 'a) -> 'a -> 'a
    val extr_matrix : 'a matrix -> int * int -> int * int -> 'a matrix
  end
```

```
module type PIXEL_MATRIX =
  sig
    type pixel
    type pixel_matrix
    val barycenter : pixel -> float
    val read_pixels : bitmapFileHeader -> bitmapInfoHeader -> in_channel ->
      pixel_matrix
    val write_pixels : out_channel -> pixel_matrix -> unit
    val read_bmp : string -> pixel_matrix
    val write_bmp : string -> pixel_matrix -> unit
    val intmatrix_of_matrix : float Matrix.matrix -> int Matrix.matrix
    val gray_levels : pixel_matrix -> float Matrix.matrix
    val pick_color : pixel -> int -> int
    val extr_uplet : pixel_matrix -> int -> float Matrix.matrix
    val red_filter : pixel_matrix -> float Matrix.matrix
    val blue_filter : pixel_matrix -> float Matrix.matrix
    val green_filter : pixel_matrix -> float Matrix.matrix
    val combine_filters : int Matrix.matrix -> int Matrix.matrix ->
      int Matrix.matrix -> pixel_matrix
  end
```

Fichier *bmp.ml*

```
type word = int
type dword = int
type bitmapFileHeader = {
  bfType : string;
  bfSize : dword;
  bfReserved1 : word;
  bfReserved2 : word;
  bfOffBits : dword;
}
type bitmapInfoHeader = {
  biSize : dword;
  biWidth : dword;
  biHeight : dword;
  biPlanes : word;
  biBitCount : word;
  biCompression : dword;
  biSizeImage : dword;
  biXPelsPerMeter : dword;
  biYPelsPerMeter : dword;
  biClrUsed : dword;
  biClrImportant : dword;
}
val read_type : in_channel -> string
val write_type : out_channel -> unit
val read_dword : in_channel -> int
val write_dword : out_channel -> int -> unit
val read_word : in_channel -> int
val write_word : out_channel -> int -> unit
val read_file_header : in_channel -> bitmapFileHeader
val write_file_header : out_channel -> bitmapFileHeader -> unit
val read_info_header : in_channel -> bitmapInfoHeader
val write_info_header : out_channel -> bitmapInfoHeader -> unit
val offset : int -> int
val make_file_header : int -> int -> bitmapFileHeader
val make_info_header : dword -> dword -> bitmapInfoHeader
```

Fichier *fwt.ml*

```
val op : float -> float
val inv : float -> float
val vect_1 : float Vect.vect
val round_2 : int -> int
val pow : int -> int -> int
val normalise : float Matrix.matrix -> int * int -> int ->
  float Matrix.matrix * (int * int)
val denormalise : 'a Matrix.matrix -> int * int -> 'a Matrix.matrix
val aco : float Vect.vect -> float Vect.vect -> float Vect.vect
val hi_up : float Vect.vect -> float Vect.vect -> float Vect.vect
val lo_conv : float Vect.vect -> float Vect.vect -> float Vect.vect
val ico : float Vect.vect -> float Vect.vect -> float Vect.vect
val hi_conv : float Vect.vect -> float Vect.vect -> float Vect.vect
val fwt : int -> float Matrix.matrix -> int -> float Vect.vect ->
  float Matrix.matrix
```

Fichier *compression_wt.ml*

```
val haar_filter : float Vect.vect
val haar_filter2 : float Vect.vect
val decompo_wt_matrix : float Matrix.matrix -> int -> float Matrix.matrix
val decompo_wt : string -> int -> float Matrix.matrix
val calc_indice_div : int -> int * int -> int * int
val calc_indice_mult : int -> int * int -> int * int
val txt_of_matrix : int Matrix.matrix -> int -> string -> int Matrix.matrix
val decompo_line : string -> int * int * int
val matrix_of_txt : float Matrix.matrix -> string -> float Matrix.matrix * int
val compresser_img : string -> int -> string -> unit
val decompresser_img : string -> string -> string -> unit
val error : string -> int -> int -> int
```