List Toolbox and Midterms

Do not use the @ operator neither any function of the `List` module.

1 Toolbox (`list_tools.ml`)

1.1 Basics

Some functions of this part are needed for next pratical!
File to submit (**and to keep!**): `list_tools.ml`.

`length`*

Write a function that computes the length of a list.

```
val length : 'a list -> int = <fun>

# length [0; 1; 0; 1; 0; 0; 0; 0; 1; 1] ;;
- : int = 10
```

`nth`*

Write a function that returns the n^{th} element of a list. The function has to raise an exception `Invalid_argument` if n is negative or an exception `Failure` if the list is too short.

```
val nth : int -> 'a list -> 'a = <fun>

# nth 5 [1; 2; 3; 4; 5] ;;
Exception: Failure "nth: list is too short".
# nth 0 ['a'; 'b'; 'c'] ;;
- : char = 'a'
# nth (-5) [] ;;
Exception: Invalid_argument "nth: index must be a natural".
```

`is_pos`

Write a function `is_pos list` that checks if all elements of list `list` are greater or equal than 0.

```
val is_pos : int list -> bool = <fun>

# is_pos [];;
- : bool = true
# is_pos [5; 8; 8; 1; 0; 10];;
- : bool = true
# is_pos [5; 8; -2; 1; 0; -1];;
- : bool = false
# is_pos [5; 8; 8; 1; 0; -1];;
- : bool = false
```

`get_max`

Write a function `get_max list` that returns the maximum element of list `list`.

```
val get_max : 'a list -> 'a = <fun>

# get_max [];;
Exception: Invalid_argument "get_max: empty list".
# get_max [5; 8; 8; 1; 0; 10];;
- : int = 10
# get_max [5; 8; 8; 1];;
- : int = 8
# get_max [10; 5; 8; 8; 1; 0];;
- : int = 10
```

1.2 Build - Modify

`init_list`

Write the function `init_list n x` that builds a list of `n` values `x`.

```
val init_list : int -> 'a -> 'a list = <fun>

# init_list 5 0 ;;
- : int list = [0; 0; 0; 0; 0]
# init_list 0 'a' ;;
- : char list = []
# init_list (-5) 1.5 ;;
Exception: Invalid_argument "init_list: n must be a natural".
```

`append`

Write the function `append` that concatenates two lists.

```
val append : 'a list -> 'a list -> 'a list = <fun>

# append [1; 2; 3] [4; 5] ;;
- : int list = [1; 2; 3; 4; 5]

# append ['a'; 'b'; 'c'] [] ;;
- : char list = ['a'; 'b'; 'c']
```

`put_list`

Write the function `put_list` *v i list* that replaces the value at position *i* in *list* by *v* when possible.

```
val put_list : 'a -> int -> 'a list -> 'a list = <fun>

# put_list 'x' 3 ['-'; '-'; '-'; '-'; '-'; '-'] ;;
- : char list = ['-'; '-'; '-'; 'x'; '-'; '-']
# put_list 0 10 [1; 1; 1; 1] ;;
- : int list = [1; 1; 1; 1]
```

1.3 'a list list

In the following, we will call *matrix* (`board`) a list of lists where all sub-lists have the same length.
See below examples of applications of the following functions.

`init_board`

Write the function `init_board` (*l*, *c*) *val* that generates a matrix of size $l \times c$ filled with *val*.

```
val init_board : int * int -> 'a -> 'a list list = <fun>
```

`is_board`

Write a function `is_board board` that checks if the list of lists `board` is a valid matrix, i.e., all sublists have the same size.

```
val is_board : 'a list list -> bool = <fun>

# is_board [];;
- : bool = true
# is_board [['*';'-';'*'];['-';'-';'*']];;
- : bool = true
# is_board [['*';'-';'*'];['*';'*']];;
- : bool = false
# is_board [['*';'*';'-'];['*';'-'];['*';'*']];;
- : bool = false
# is_board [['*';'*'];['-';'-'];['*';'*';'*']];;
- : bool = false
```

`print_board`

Write a function `print_board board` that prints the matrix `board`. It displays all element of a line successively and the last element of a line is followed by a line break.

```
val print_board : char list list -> unit = <fun>

# print_board (init_board (2,2) '*');;
**
**
- : unit = ()
# print_board (init_board (5,5) '-');;
-----
-----
-----
-----
-----
- : unit = ()
```

`get_cell`

Write the function `get_cell` $(x,\ y)$ *board* that returns the value at position $(x,\ y)$ in the matrix *board*.

```
val get_cell : int * int -> 'a list list -> 'a = <fun>
```

`put_cell`

Write the function `put_cell` *val* $(x,\ y)$ *board* that replaces the value at $(x,\ y)$ in *board* by the value *val*. If the cell $(x,\ y)$ does not exist, *board* is returned unchanged (no exception).

```
val put_cell : 'a -> int * int -> 'a list list -> 'a list list = <fun>
```

Examples

```
# let board = init_board (5, 3) 0;;
val board : int list list =
  [[0; 0; 0]; [0; 0; 0]; [0; 0; 0]; [0; 0; 0]; [0; 0; 0]]

# let board = put_cell 1 (0, 0) board;;
val board : int list list =
  [[1; 0; 0]; [0; 0; 0]; [0; 0; 0]; [0; 0; 0]; [0; 0; 0]]

# let board = put_cell 2 (2, 1) board;;
val board : int list list =
  [[1; 0; 0]; [0; 0; 0]; [0; 2; 0]; [0; 0; 0]; [0; 0; 0]]

# get_cell (2, 1) board;;
- : int = 2
```

2 Midterms...

2.1 File loading

The directive *#use* (with the #) can be used directly in the toplevel to load functions from previous files more easily:

```
#use "file-name";;

Read, compile and execute source phrases from the given file. This is textual
inclusion:
phrases are processed just as if they were typed on standard input. The reading
of the file stops at the first error encountered.
```

For example,

```
#use "list_tools.ml";;
```

will load all the definitions of section 1.1 in the CAML environment

Exercise 1: Histogram sort (`histo.ml`)

Histogram sort principle:

The histogram sort is a sorting algorithm that can only be applied on natural integers.

The histogram `hist` of a list `list` is defined as follows:

- `hist` is a list of size `m+1` where `m` is the maximum value of the list `list`
- the ithvalue `x` of `hist` corresponds to the occurence number (number of times) of the element `x` in the list `list`

Example:

```
# list;;
- : int list = [1; 5; 0; 9; 4; 4; 3]
# hist;;
- : int list = [1; 1; 0; 1; 2; 1; 0; 0; 0; 1]
```

- `hist` is of size 10 (9+1) since the maximum value of `list` is 9.
- The first value (index 0) of `hist` is 1 since 0 is present once in the list `list`.
- The second value (index 1) of `hist` is 1 since 1 is present once in the list `list`.
- The third value (index 2) of `hist` is 0 since 2 is not present in the list `list`.

The list `list` can be sorted from this histogram `hist` by creating a new list `sorted_list`. The histogram `hist` is scanned from index 0 to the last index. For each value `x` of histogram `hist` at index `i`, the element `i` is added `x` times in the list `sorted_list`.

add_occ

Write a function `add_occ i hist` that adds 1 to the value of i^{th}element in histogram `hist`.

```
val add_occ : int -> int list -> int list = <fun>

# add_occ (-1) [0; 0; 2; 5; 0; 1; 1; 0];;
Exception: Invalid_argument "add_occ: i should be >=0".
# add_occ 10 [0; 0; 2; 5; 0; 1; 1; 0];;
Exception: Invalid_argument "add_occ: hist too short".
# add_occ 0 [0; 0; 2; 5; 0; 1; 1; 0];;
- : int list = [1; 0; 2; 5; 0; 1; 1; 0]
# add_occ 4 [0; 0; 2; 5; 0; 1; 1; 0];;
- : int list = [0; 0; 2; 5; 1; 1; 1; 0]
# add_occ 5 [0; 0; 2; 5; 0; 1; 1; 0];;
# add_occ 2 [];;
- : int list = [0; 0; 2; 5; 0; 2; 1; 0]
Exception: Invalid_argument "add_occ: hist too short".
```

get_hist

Write a function `get_hist list` that builds the histogram associated to the list `list` which only contains naturals.

```
val get_hist : int list -> int list = <fun>

# get_hist [0; 1; 2; 3]
- : int list = [1; 1; 1; 1]
# get_hist [5; 5; 5; 5];;
- : int list = [0; 0; 0; 0; 0; 4]
# get_hist [1; 5; 0; 9; 4; 4; 3];;
- : int list = [1; 1; 0; 1; 2; 1; 0; 0; 0; 1]
# get_hist [10; 5; 8; 8; 1; 0];;
- : int list = [1; 1; 0; 0; 0; 1; 0; 0; 2; 0; 1]
```

get_sorted

Write a function `get_sorted list` that builds the sorted list from the supposed valid histogram `hist`.

```
val get_sorted : int list -> int list = <fun>

# get_sorted [0; 1; 1; 1];;
- : int list = [1; 2; 3]
# get_sorted [0; 0; 0; 2; 0; 4];;
- : int list = [3; 3; 5; 5; 5; 5; 5]
# get_sorted [1; 1; 0; 1; 2; 1; 0; 0; 0; 1];;
- : int list = [0; 1; 3; 4; 4; 5; 9]
# get_sorted [1; 1; 0; 0; 0; 1; 0; 0; 2; 0; 1];;
- : int list = [0; 1; 5; 8; 8; 10]
```

hist_sort

Write a function `hist_sort list` that sorts the list `list` by building and returning a new list thanks to the histogram sort.

```
val hist_sort : int list -> int list = <fun>

# hist_sort [0; 5; 4; -1; 8; 3] ;;
Exception: Invalid_argument "hist_sort: I cannot sort that list".
# hist_sort [] ;;
- : a list = []
# hist_sort [12; 150; 66; 0; 12; 88; 5; 12; 555; 5; 1; 150] ;;
- : int list = [0; 1; 5; 5; 12; 12; 12; 66; 88; 150; 150; 555]
# hist_sort [1; 5; 0; 9; 4; 4; 3];;
- : int list = [0; 1; 3; 4; 4; 5; 9]
# hist_sort [10; 5; 8; 8; 1; 0];;
- : int list = [0; 1; 5; 8; 8; 10]
```

Exercise 2: Patterns and matrices (matrix_patterns.ml)

The goal of this exercise is to draw patterns inside matrices. Matrices will be represented as lists of lists in Caml. Each line of the matrix is represented by a sublist.
A matrix of size `n*m`:

- has `n` lines of size m
- has `m` lines of size n
- is *square* if `n = m`

Examples:

```
# mat1;;
- : char list = [['*'; '*'; '-']; ['*'; '-'; '-']; ['*'; '*'; '*']]
# mat2;;
- : char list = [['-']; ['*']; ['*']]
# not_a_mat1;;
- : char list = [['*'; '*'; '-']; ['*'; '-'; '-']; ['*'; '*']]
# not_a_mat2;;
- : char list = [['*'; '*'; '-']; ['*'; '*']; ['*']]
```

The list of lists `mat1` is a matrix of size `3*3`.
The list of lists `mat2` is a matrix of size `3*1`.
The list of lists `not_a_mat1` is not a matrix since the size of the last line (2) is smaller than the size of the two first lines (3).
The list of lists `not_a_mat2` is not a matrix since all the lines have different sizes.

With the exception of the function `is_mat`, all matrices given as parameters are assumed valid (all sublists have the same size).

The functions `print_char` and `print_newline` can be used, remainder:

```
# print_char;; (*prints a character*)
- : char -> unit = <fun>
# print_newline;; (*prints the newline character*)
- : unit -> unit = <fun>
# print_char '*';;
*- : unit = ()
# print_char '*';print_newline ();;
*
- : unit = ()
```

mat_cross

Write the function `mat_cross n c1 c2` that builds the following pattern as a matrix for every odd integer `n` greater than 1. The resulting matrix is a square matrix. The elements for which one of the coordinate of their position is equal to `n/2` have the value `c1`, the others have the value `c2`.

```
val mat_cross : int -> 'a -> 'a -> 'a list list = <fun>

# mat_cross 3 '*' '-';;
- : char list list = [['-'; '*'; '-']; ['*'; '*'; '*']; ['-'; '*'; '-']]
# mat_cross 1 '*' '-';;
Exception: Invalid_argument "mat_cross: n should be odd and >1".
# mat_cross 6 '*' '-';;
Exception: Invalid_argument "mat_cross: n should be odd and >1".
# print_board (mat_cross 5 '*' '-');;
--*--
--*--
*****
--*--
--*--
- : unit = ()
```

mat_cross_diag

Write the function `mat_cross_diag n c1 c2` that builds the following pattern as a matrix for every integer `n` greater than 1. The resulting matrix is a square matrix of size `n*n`. The elements which positions belong to one of the two diagonals of the matrix have the value `c1`, the others have the value `c2`.

```
val mat_cross_diag : int -> 'a -> 'a -> 'a list list = <fun>

# mat_cross_diag 3 '*' '-';;
- : char list list = [['*'; '-'; '*']; ['-'; '*'; '-']; ['*'; '-'; '*']]
# mat_cross_diag 1 '*' '-';;
Exception: Invalid_argument "mat_cross_diag: n must be >1".
# print_board (mat_cross_diag 5 '*' '-');;
*---*
-*-*-
--*--
-*-*-
*---*
- : unit = ()
# print_board (mat_cross_diag 6 '*' '-');;
*----*
-*--*-
--**--
--**--
-*--*-
*----*
- : unit = ()
```

mat_square

Write the function `mat_square n c1 c2` that builds the following pattern as a matrix for every integer `n` greater than 1. The resulting matrix is a square matrix of size `n*n`. The first line starts with the pattern `c1`, then the pattern `c2` and continues alternating between `c1` and `c2` till the end of the line. The second line follows the same pattern but this time starting by the pattern `c2`. The third line is equal to the first line and the following lines follow the same pattern.

```
val mat_square : int -> 'a -> 'a -> 'a list list = <fun>

# mat_square 2 '*' '.';;
- : char list list = [['*'; '.']; ['.'; '*']]
# mat_square 1 '*' '.';;
Exception: Invalid_argument "mat_square: n must be >1">
# print_board (mat_square 1 '*' '.');;
Exception: Invalid_argument "mat_square: n must be >1".
# print_board (mat_square 5 '*' '.');;
*.*.*
.*.*.
*.*.*
.*.*.
*.*.*
- : unit = ()
# print_board (mat_square 6 '*' '.');;
*.*.*.
.*.*.*
*.*.*.
.*.*.*
*.*.*.
.*.*.*
- : unit = ()
```

mat_pattern_batch

Write the function `mat_pattern_batch coor mat` that takes as parameters a matrix `mat` and a triplets list `coor`. It returns `mat` in which for each triplet `(x,y,v)` of `coor` the value in position `(x,y)` has been replaced by `v`.

```
  val mat_pattern_batch : (int * int * 'a) list -> 'a list list -> 'a list list =
<fun>

  # print_board (mat_pattern_batch [(1,2,'a');(4,1,'b')] (init_board (5,5) '*'));;
  *****
  **a**
  *****
  *****
  *b***
  - : unit = ()
```