Module type Map.S

`module type S = sig .. end`

Output signature of the functor `Map.Make`.

`type key`

The type of the map keys.

`type +'a t`

The type of maps from type `key` to type `'a`.

`val empty : 'a t`

The empty map.

`val is_empty : 'a t -> bool`

Test whether a map is empty or not.

`val mem : key -> 'a t -> bool`

`mem x m` returns `true` if `m` contains a binding for `x`, and `false` otherwise.

`val add : key -> 'a -> 'a t -> 'a t`

`add x y m` returns a map containing the same bindings as `m`, plus a binding of `x` to `y`. If `x` was already bound in `m` to a value that is physically equal to `y`, `m` is returned unchanged (the result of the function is then physically equal to `m`). Otherwise, the previous binding of `x` in `m` disappears.
Before 4.03 Physical equality was not ensured.

`val update : key -> ('a option -> 'a option) -> 'a t -> 'a t`

`update x f m` returns a map containing the same bindings as `m`, except for the binding of `x`. Depending on the value of `y` where `y` is `f (find_opt x m)`, the binding of `x` is added, removed or updated. If `y` is `None`, the binding is removed if it exists; otherwise, if `y` is `Some z` then `x` is associated to `z` in the resulting map. If `x` was already bound in `m` to a value that is physically equal to `z`, `m` is returned unchanged (the result of the function is then physically equal to `m`).
Since 4.06.0

`val singleton : key -> 'a -> 'a t`

`singleton x y` returns the one-element map that contains a binding `y` for `x`.
Since 3.12.0

`val remove : key -> 'a t -> 'a t`

`remove x m` returns a map containing the same bindings as `m`, except for `x` which is unbound in the returned map. If `x` was not in `m`, `m` is returned unchanged (the result of the function is then physically equal to `m`).
Before 4.03 Physical equality was not ensured.

```
val merge : (key -> 'a option -> 'b option -> 'c option) ->
       'a t -> 'b t -> 'c t
```

`merge f m1 m2` computes a map whose keys is a subset of keys of `m1` and of `m2`. The presence of each such binding, and the corresponding value, is determined with the function `f`. In terms of the `find_opt` operation, we have `find_opt x (merge f m1 m2) = f (find_opt x m1) (find_opt x m2)` for any key `x`, provided that `f None None = None`.
Since 3.12.0

```
val union : (key -> 'a -> 'a -> 'a option) ->
       'a t -> 'a t -> 'a t
```

`union f m1 m2` computes a map whose keys is the union of keys of `m1` and of `m2`. When the same binding is defined in both arguments, the function `f` is used to combine them. This is a special case of `merge`: `union f m1 m2` is equivalent to `merge f' m1 m2`, where

- `f' None None = None`
- `f' (Some v) None = Some v`
- `f' None (Some v) = Some v`
- `f' (Some v1) (Some v2) = f v1 v2`

Since 4.03.0

```
val compare : ('a -> 'a -> int) -> 'a t -> 'a t -> int
```

Total ordering between maps. The first argument is a total ordering used to compare data associated with equal keys in the two maps.

```
val equal : ('a -> 'a -> bool) -> 'a t -> 'a t -> bool
```

`equal cmp m1 m2` tests whether the maps `m1` and `m2` are equal, that is, contain equal keys and associate them with equal data. `cmp` is the equality predicate used to compare the data associated with the keys.

```
val iter : (key -> 'a -> unit) -> 'a t -> unit
```

`iter f m` applies `f` to all bindings in map `m`. `f` receives the key as first argument, and the associated value as second argument. The bindings are passed to `f` in increasing order with respect to the ordering over the type of the keys.

```
val fold : (key -> 'a -> 'b -> 'b) -> 'a t -> 'b -> 'b
```

`fold f m a` computes `(f kN dN ... (f k1 d1 a)...)`, where `k1 ... kN` are the keys of all bindings in `m` (in increasing order), and `d1 ... dN` are the associated data.

```
val for_all : (key -> 'a -> bool) -> 'a t -> bool
```

`for_all p m` checks if all the bindings of the map satisfy the predicate `p`.
Since 3.12.0

```
val exists : (key -> 'a -> bool) -> 'a t -> bool
```

`exists p m` checks if at least one binding of the map satisfies the predicate `p`.
Since 3.12.0

```
val filter : (key -> 'a -> bool) -> 'a t -> 'a t
```

`filter p m` returns the map with all the bindings in `m` that satisfy predicate `p`. If `p` satisfies every binding in `m`, `m` is returned unchanged (the result of the function is then physically equal to `m`)
Before 4.03 Physical equality was not ensured.
Since 3.12.0

```
val partition : (key -> 'a -> bool) -> 'a t -> 'a t * 'a t
```

`partition p m` returns a pair of maps `(m1, m2)`, where `m1` contains all the bindings of `s` that satisfy the predicate `p`, and `m2` is the map with all the bindings of `s` that do not satisfy `p`.
Since 3.12.0

```
val cardinal : 'a t -> int
```

Return the number of bindings of a map.
Since 3.12.0

```
val bindings : 'a t -> (key * 'a) list
```

Return the list of all bindings of the given map. The returned list is sorted in increasing order with respect to the ordering `Ord.compare`, where `Ord` is the argument given to `Map.Make`.

Since 3.12.0

`val min_binding : 'a t -> key * 'a`

Return the smallest binding of the given map (with respect to the `Ord.compare` ordering), or raise `Not_found` if the map is empty.
Since 3.12.0

`val min_binding_opt : 'a t -> (key * 'a) option`

Return the smallest binding of the given map (with respect to the `Ord.compare` ordering), or `None` if the map is empty.
Since 4.05

`val max_binding : 'a t -> key * 'a`

Same as `Map.S.min_binding`, but returns the largest binding of the given map.
Since 3.12.0

`val max_binding_opt : 'a t -> (key * 'a) option`

Same as `Map.S.min_binding_opt`, but returns the largest binding of the given map.
Since 4.05

`val choose : 'a t -> key * 'a`

Return one binding of the given map, or raise `Not_found` if the map is empty. Which binding is chosen is unspecified, but equal bindings will be chosen for equal maps.
Since 3.12.0

`val choose_opt : 'a t -> (key * 'a) option`

Return one binding of the given map, or `None` if the map is empty. Which binding is chosen is unspecified, but equal bindings will be chosen for equal maps.
Since 4.05

`val split : key -> 'a t -> 'a t * 'a option * 'a t`

`split x m` returns a triple `(l, data, r)`, where `l` is the map with all the bindings of `m` whose key is strictly less than `x`; `r` is the map with all the bindings of `m` whose key is strictly greater than `x`; `data` is `None` if `m` contains no binding for `x`, or `Some v` if `m` binds `v` to `x`.
Since 3.12.0

`val find : key -> 'a t -> 'a`

`find x m` returns the current binding of `x` in `m`, or raises `Not_found` if no such binding exists.

`val find_opt : key -> 'a t -> 'a option`

`find_opt x m` returns `Some v` if the current binding of `x` in `m` is `v`, or `None` if no such binding exists.
Since 4.05

`val find_first : (key -> bool) -> 'a t -> key * 'a`

`find_first f m`, where `f` is a monotonically increasing function, returns the binding of `m` with the lowest key `k` such that `f k`, or raises `Not_found` if no such key exists.

For example, `find_first (fun k -> Ord.compare k x >= 0) m` will return the first binding `k, v` of `m` where `Ord.compare k x >= 0` (intuitively: `k >= x`), or raise `Not_found` if `x` is greater than any element of `m`.

Since 4.05

`val find_first_opt : (key -> bool) -> 'a t -> (key * 'a) option`

`find_first_opt f m`, where `f` is a monotonically increasing function, returns an option containing the binding of `m` with the lowest key `k` such that `f k`, or `None` if no such key exists.
Since 4.05

```
val find_last : (key -> bool) -> 'a t -> key * 'a
```

`find_last f m`, where `f` is a monotonically decreasing function, returns the binding of `m` with the highest key `k` such that `f k`, or raises `Not_found` if no such key exists.
Since 4.05

```
val find_last_opt : (key -> bool) -> 'a t -> (key * 'a) option
```

`find_last_opt f m`, where `f` is a monotonically decreasing function, returns an option containing the binding of `m` with the highest key `k` such that `f k`, or `None` if no such key exists.
Since 4.05

```
val map : ('a -> 'b) -> 'a t -> 'b t
```

`map f m` returns a map with same domain as `m`, where the associated value `a` of all bindings of `m` has been replaced by the result of the application of `f` to `a`. The bindings are passed to `f` in increasing order with respect to the ordering over the type of the keys.

```
val mapi : (key -> 'a -> 'b) -> 'a t -> 'b t
```

Same as `Map.S.map`, but the function receives as arguments both the key and the associated value for each binding of the map.

Iterators

```
val to_seq : 'a t -> (key * 'a) Seq.t
```

Iterate on the whole map, in ascending order
Since 4.07

```
val to_seq_from : key -> 'a t -> (key * 'a) Seq.t
```

`to_seq_from k m` iterates on a subset of the bindings of `m`, in ascending order, from key `k` or above.
Since 4.07

```
val add_seq : (key * 'a) Seq.t -> 'a t -> 'a t
```

Add the given bindings to the map, in order.
Since 4.07

```
val of_seq : (key * 'a) Seq.t -> 'a t
```

Build a map from the given bindings
Since 4.07