1. Hash tables for hash consing.

The technique is described in this paper: Sylvain Conchon and Jean-Christophe FilliÃtre. Type-Safe Modular Hash-Consing. In ACM SIGPLAN Workshop on ML, Portland, Oregon, September 2006. https://www.lri.fr/ filliatr/ftp/publis/hash-consing2.pdf

Note: a different, more elaborated hash-consing library can be found in Why3 sources at http://why3.lri.fr/

Hash consed values are of the following type *hash_consed*. The field *tag* contains a unique integer (for values hash consed with the same table). The field *hkey* contains the hash key of the value (without modulo) for possible use in other hash tables (and internally when hash consing tables are resized). The field *node* contains the value itself.

Hash consing tables are using weak pointers, so that values that are no more referenced from anywhere else can be erased by the GC.

```
type +α hash_consed = private {
  hkey : int;
  tag : int;
  node : α }
```

2. Generic part, using ocaml generic equality and hash function.

```
type α t

val create : int → α t
  (** create n creates an empty table of initial size n. The table will grow as needed. *)
val clear : α t → unit
  (** Removes all elements from the table. *)
val hashcons : α t → α → α hash_consed
  (** hashcons t n hash-cons the value n using table t i.e. returns any existing value in t
  equal to n, if any; otherwise, allocates a new one hash-consed value of node n and returns
  it. As a consequence the returned value is physically equal to any equal value already hash-
  consed using table t. *)
val iter : (α hash_consed → unit) → α t → unit
  (** iter f t iterates f over all elements of t. *)
val stats : α t → int × int × int × int × int × int
  (** Return statistics on the table. The numbers are, in order: table length, number of
  entries, sum of bucket lengths, smallest bucket length, median bucket length, biggest bucket
  length. *)
```

3. Functorial interface.

```
module type HashedType =
  sig
    type t
    val equal : t -> t -> bool
    val hash : t -> int
  end

module type S =
  sig
    type key
    type t
    val create : int -> t
    val clear : t -> unit
    val hashcons : t -> key -> key hash_consed
    val iter : (key hash_consed -> unit) -> t -> unit
    val stats : t -> int * int * int * int * int * int
  end

module Make(H : HashedType) : (S with type key = H.t)
```