Interface for module Size

1. Sizes of ocaml values (in their memory representation). Sizes are given in words (*size_w*), bytes (*size_b*) or kilobytes (*size_kb*), in a system-independent way.

```
val size_w : α → int
val size_b : α → int
val size_kb : α → int
```

Module Size

2. Pointers already visited are stored in a hash-table, where comparisons are done using physical equality.

```
module H = Hashtbl.Make(
  struct
    type t = Obj.t
    let equal = (≡)
    let hash o = Hashtbl.hash (magic o : int)
  end)

let node_table = (H.create 257 : unit H.t)

let in_table o = try H.find node_table o; true with Not_found → false

let add_in_table o = H.add node_table o ()

let reset_table () = H.clear node_table
```

3. Objects are traversed recursively, as soon as their tags are less than *no_scan_tag*. *count* records the numbers of words already visited.

```
let size_of_double = size (repr 1.0)

let count = ref 0
```

```
let rec traverse t =
  if ¬ (in_table t) then begin
    add_in_table t;
    if is_block t then begin
      let n = size t in
      let tag = tag t in
      if tag < no_scan_tag then begin
        count := !count + 1 + n;
        for i = 0 to n − 1 do
          let f = field t i in
          if is_block f then traverse f
        done
      end else if tag = string_tag then
        count := !count + 1 + n
      else if tag = double_tag then
        count := !count + size_of_double
      else if tag = double_array_tag then
        count := !count + 1 + size_of_double × n
      else
        incr count
    end
  end
```

4. Sizes of objects in words and in bytes. The size in bytes is computed system-independently according to *Sys.word_size*.

```
let size_w o =
  reset_table ();
  count := 0;
  traverse (repr o);
  !count

let size_b o = (size_w o) × (Sys.word_size / 8)

let size_kb o = (size_w o) / (8192 / Sys.word_size)
```

Index