GDSL Reference Manual

1.3

Generated by Doxygen 1.3.5

Sun Oct 3 16:15:50 2004

Contents

Chapter 1

GDSL Module Index

1.1 GDSL Modules

Here is a list of all modules:

Chapter 2

GDSL File Index

2.1 GDSL File List

Here is a list of all files with brief descriptions:

Chapter 3

GDSL Module Documentation

3.1 Low level binary tree manipulation module

Typedefs

- typedef _gdsl_bintree * **_gdsl_bintree_t**

  GDSL low-level binary tree type.

- typedef int(* **gdsl_bintree_map_func_t**)(**_gdsl_bintree_t** TREE, void *USER_DATA)

  GDSL low-level binary tree map function type.

Functions

- **_gdsl_bintree_t _gdsl_bintree_alloc** (const **gdsl_element_t** E, const **_gdsl_bintree_t** LEFT, const **_gdsl_bintree_t** RIGHT)

  Create a new low-level binary tree.

- void **_gdsl_bintree_free** (**_gdsl_bintree_t** T, const **gdsl_free_-func_t** FREE_F)

  Destroy a low-level binary tree.

- **_gdsl_bintree_t _gdsl_bintree_copy** (const **_gdsl_bintree_t** T, const **gdsl_copy_func_t** COPY_F)

  Copy a low-level binary tree.

- **bool _gdsl_bintree_is_empty** (const **_gdsl_bintree_t** T)

  Check if a low-level binary tree is empty.

- **bool _gdsl_bintree_is_leaf** (const **_gdsl_bintree_t** T)

  Check if a low-level binary tree is reduced to a leaf.

- **bool _gdsl_bintree_is_root** (const **_gdsl_bintree_t** T)

  Check if a low-level binary tree is a root.

- **gdsl_element_t _gdsl_bintree_get_content** (const **_gdsl_bintree_t** T)

  Get the root content of a low-level binary tree.

- **_gdsl_bintree_t _gdsl_bintree_get_parent** (const **_gdsl_bintree_t** T)

  Get the parent tree of a low-level binary tree.

- **_gdsl_bintree_t _gdsl_bintree_get_left** (const **_gdsl_bintree_t** T)

  Get the left sub-tree of a low-level binary tree.

- **_gdsl_bintree_t _gdsl_bintree_get_right** (const **_gdsl_bintree_t** T)

  Get the right sub-tree of a low-level binary tree.

- **_gdsl_bintree_t** * **_gdsl_bintree_get_left_ref** (const **_gdsl_bintree_t** T)

  Get the left sub-tree reference of a low-level binary tree.

- **_gdsl_bintree_t** * **_gdsl_bintree_get_right_ref** (const **_gdsl_bintree_t** T)

  Get the right sub-tree reference of a low-level binary tree.

- **ulong _gdsl_bintree_get_height** (const **_gdsl_bintree_t** T)

  Get the height of a low-level binary tree.

- **ulong _gdsl_bintree_get_size** (const **_gdsl_bintree_t** T)

  Get the size of a low-level binary tree.

- void **_gdsl_bintree_set_content** (**_gdsl_bintree_t** T, const **gdsl_element_t** E)

  Set the root element of a low-level binary tree.

- void **_gdsl_bintree_set_parent** (**_gdsl_bintree_t** T, const **_gdsl_bintree_t** P)

  Set the parent tree of a low-level binary tree.

- void **_gdsl_bintree_set_left** (**_gdsl_bintree_t** T, const **_gdsl_bintree_t** L)

Set left sub-tree of a low-level binary tree.

- void **_gdsl_bintree_set_right** (**_gdsl_bintree_t** T, const **_gdsl_bintree_t** R)

  Set right sub-tree of a low-level binary tree.

- **_gdsl_bintree_t _gdsl_bintree_rotate_left** (**_gdsl_bintree_t** *T)

  Left rotate a low-level binary tree.

- **_gdsl_bintree_t _gdsl_bintree_rotate_right** (**_gdsl_bintree_t** *T)

  Right rotate a low-level binary tree.

- **_gdsl_bintree_t _gdsl_bintree_rotate_left_right** (**_gdsl_bintree_t** *T)

  Left-right rotate a low-level binary tree.

- **_gdsl_bintree_t _gdsl_bintree_rotate_right_left** (**_gdsl_bintree_t** *T)

  Right-left rotate a low-level binary tree.

- **_gdsl_bintree_t _gdsl_bintree_map_prefix** (const **_gdsl_bintree_t** T, const **gdsl_bintree_map_func_t** MAP_F, void *USER_DATA)

  Parse a low-level binary tree in prefixed order.

- **_gdsl_bintree_t _gdsl_bintree_map_infix** (const **_gdsl_bintree_t** T, const **gdsl_bintree_map_func_t** MAP_F, void *USER_DATA)

  Parse a low-level binary tree in infixed order.

- **_gdsl_bintree_t _gdsl_bintree_map_postfix** (const **_gdsl_bintree_t** T, const **gdsl_bintree_map_func_t** MAP_F, void *USER_DATA)

  Parse a low-level binary tree in postfixed order.

- void **_gdsl_bintree_write** (const **_gdsl_bintree_t** T, const **gdsl_write_func_t** WRITE_F, FILE *OUTPUT_FILE, void *USER_DATA)

  Write the content of all nodes of a low-level binary tree to a file.

- void **_gdsl_bintree_write_xml** (const **_gdsl_bintree_t** T, const **gdsl_write_func_t** WRITE_F, FILE *OUTPUT_FILE, void *USER_DATA)

  Write the content of a low-level binary tree to a file into XML.

- void **_gdsl_bintree_dump** (const **_gdsl_bintree_t** T, const **gdsl_write_func_t** WRITE_F, FILE *OUTPUT_FILE, void *USER_DATA)

  Dump the internal structure of a low-level binary tree to a file.

3.1.1 Typedef Documentation

3.1.1.1 typedef struct _gdsl_bintree* _gdsl_bintree_t

GDSL low-level binary tree type.

This type is voluntary opaque. Variables of this kind could'nt be directly used, but by the functions of this module.

Definition at line 53 of file _gdsl_bintree.h.

3.1.1.2 typedef int(* gdsl_bintree_map_func_t)(_gdsl_bintree_t TREE, void *USER_DATA)

GDSL low-level binary tree map function type.

Parameters:

TREE The low-level binary tree to map.

USER_DATA The user datas to pass to this function.

Returns:

0 if the mapping must be stopped, another value otherwise.

Definition at line 61 of file _gdsl_bintree.h.

3.1.2 Function Documentation

3.1.2.1 _gdsl_bintree_t _gdsl_bintree_alloc (const gdsl_element_t *E*, const _gdsl_bintree_t *LEFT*, const _gdsl_bintree_t *RIGHT*)

Create a new low-level binary tree.

Allocate a new low-level binary tree data structure. Its root content is set to E and its left son (resp. right) is set to LEFT (resp. RIGHT).

Note:

Complexity: O(1)

Precondition:

nothing.

Parameters:

E The root content of the new low-level binary tree to create.

LEFT The left sub-tree of the new low-level binary tree to create.

RIGHT The right sub-tree of the new low-level binary tree to create.

Returns:

the newly allocated low-level binary tree in case of success.
NULL in case of insufficient memory.

See also:

_gdsl_bintree_free()(p. 10)

3.1.2.2 _gdsl_bintree_t _gdsl_bintree_copy (const _gdsl_bintree_t *T*, const gdsl_copy_func_t *COPY_F*)

Copy a low-level binary tree.

Create and return a copy of the low-level binary tree T using COPY_F on each T's element to copy them.

Note:

Complexity: O(|T|)

Precondition:

COPY_F != NULL

Parameters:

T The low-level binary tree to copy.

COPY_F The function used to copy T's nodes contents.

Returns:

a copy of T in case of success.
NULL if _gdsl_bintree_is_empty (T) == TRUE or in case of insufficient memory.

See also:

_gdsl_bintree_alloc()(p. 8)
_gdsl_bintree_free()(p. 10)
_gdsl_bintree_is_empty()(p. 14)

3.1.2.3 void _gdsl_bintree_dump (const _gdsl_bintree_t *T*, const gdsl_write_func_t *WRITE_F*, FILE * *OUTPUT_FILE*, void * *USER_DATA*)

Dump the internal structure of a low-level binary tree to a file.

Dump the structure of the low-level binary tree T to OUTPUT_FILE. If WRITE_F != NULL, then use WRITE_F function to write T's nodes contents to OUTPUT_FILE. Additionnal USER_DATA argument could be passed to WRITE_F.

Note:
Complexity: O(|T|)

Precondition:
OUTPUT_FILE != NULL

Parameters:
T The low-level binary tree to dump.

WRITE_F The write function.

OUTPUT_FILE The file where to write T's nodes.

USER_DATA User's datas passed to WRITE_F.

See also:
_gdsl_bintree_write()(p. 21)
_gdsl_bintree_write_xml()(p. 22)

3.1.2.4 void _gdsl_bintree_free (_gdsl_bintree_t *T*, const gdsl_free_func_t *FREE_F*)

Destroy a low-level binary tree.

Flush and destroy the low-level binary tree T. If FREE_F != NULL, FREE_F function is used to deallocate each T's element. Otherwise nothing is done with T's elements.

Note:
Complexity: O(|T|)

Precondition:
nothing.

Parameters:
T The low-level binary tree to destroy.

FREE_F The function used to deallocate T's nodes contents.

See also:
_gdsl_bintree_alloc()(p. 8)

3.1.2.5 gdsl_element_t _gdsl_bintree_get_content (const _gdsl_bintree_t *T*)

Get the root content of a low-level binary tree.

Note:
Complexity: O(1)

Precondition:
T must be a non-empty _gdsl_bintree_t.

Parameters:
T The low-level binary tree to use.

Returns:
the root's content of the low-level binary tree T.

See also:
_gdsl_bintree_set_content()(p. 19)

3.1.2.6 ulong _gdsl_bintree_get_height (const _gdsl_bintree_t *T*)

Get the height of a low-level binary tree.

Compute the height of the low-level binary tree T (noted h(T)).

Note:
Complexity: O(|T|)

Precondition:
nothing.

Parameters:
T The low-level binary tree to use.

Returns:
the height of T.

See also:
_gdsl_bintree_get_size()(p. 14)

3.1.2.7 _gdsl_bintree_t _gdsl_bintree_get_left (const _gdsl_bintree_t *T*)

Get the left sub-tree of a low-level binary tree.

Return the left subtree of the low-level binary tree T (noted l(T)).

Note:
Complexity: O(1)

Precondition:
T must be a non-empty _gdsl_bintree_t.

Parameters:
T The low-level binary tree to use.

Returns:
the left sub-tree of the low-level binary tree T if T has a left sub-tree.
NULL if the low-level binary tree T has no left sub-tree.

See also:
_gdsl_bintree_get_right()(p. 13)
_gdsl_bintree_set_left()(p. 20)
_gdsl_bintree_set_right()(p. 21)

3.1.2.8 _gdsl_bintree_t* _gdsl_bintree_get_left_ref (const _gdsl_bintree_t *T*)

Get the left sub-tree reference of a low-level binary tree.

Note:
Complexity: O(1)

Precondition:
T must be a non-empty _gdsl_bintree_t.

Parameters:
T The low-level binary tree to use.

Returns:
the left sub-tree reference of the low-level binary tree T.

See also:
_gdsl_bintree_get_right_ref()(p. 13)

3.1.2.9 _gdsl_bintree_t _gdsl_bintree_get_parent (const _gdsl_bintree_t *T*)

Get the parent tree of a low-level binary tree.

Note:
Complexity: O(1)

Precondition:
T must be a non-empty _gdsl_bintree_t.

Parameters:
T The low-level binary tree to use.

Returns:
the parent of the low-level binary tree T if T isn't a root.
NULL if the low-level binary tree T is a root (ie. T has no parent).

See also:
_gdsl_bintree_is_root()(p. 15)
_gdsl_bintree_set_parent()(p. 20)

3.1.2.10 _gdsl_bintree_t _gdsl_bintree_get_right (const _gdsl_bintree_t *T*)

Get the right sub-tree of a low-level binary tree.

Return the right subtree of the low-level binary tree T (noted r(T)).

Note:
Complexity: O(1)

Precondition:
T must be a non-empty _gdsl_bintree_t

Parameters:
T The low-level binary tree to use.

Returns:
the right sub-tree of the low-level binary tree T if T has a right sub-tree.
NULL if the low-level binary tree T has no right sub-tree.

See also:
_gdsl_bintree_get_left()(p. 11)
_gdsl_bintree_set_left()(p. 20)
_gdsl_bintree_set_right()(p. 21)

3.1.2.11 _gdsl_bintree_t* _gdsl_bintree_get_right_ref (const _gdsl_bintree_t *T*)

Get the right sub-tree reference of a low-level binary tree.

Note:
Complexity: O(1)

Precondition:
T must be a non-empty _gdsl_bintree_t.

Parameters:
T The low-level binary tree to use.

Returns:
the right sub-tree reference of the low-level binary tree T.

See also:
_gdsl_bintree_get_left_ref()(p. 12)

3.1.2.12 ulong _gdsl_bintree_get_size (const _gdsl_bintree_t *T*)

Get the size of a low-level binary tree.

Note:
Complexity: O(|T|)

Precondition:
nothing.

Parameters:
T The low-level binary tree to use.

Returns:
the number of elements of T (noted |T|).

See also:
_gdsl_bintree_get_height()(p. 11)

3.1.2.13 bool _gdsl_bintree_is_empty (const _gdsl_bintree_t *T*)

Check if a low-level binary tree is empty.

Note:
Complexity: O(1)

Precondition:
nothing.

Parameters:
T The low-level binary tree to check.

Returns:
TRUE if the low-level binary tree T is empty.
FALSE if the low-level binary tree T is not empty.

See also:
_gdsl_bintree_is_leaf()(p. 15)
_gdsl_bintree_is_root()(p. 15)

3.1.2.14 bool _gdsl_bintree_is_leaf (const _gdsl_bintree_t *T*)

Check if a low-level binary tree is reduced to a leaf.

Note:
Complexity: O(1)

Precondition:
T must be a non-empty _gdsl_bintree_t.

Parameters:
T The low-level binary tree to check.

Returns:
TRUE if the low-level binary tree T is a leaf.
FALSE if the low-level binary tree T is not a leaf.

See also:
_gdsl_bintree_is_empty()(p. 14)
_gdsl_bintree_is_root()(p. 15)

3.1.2.15 bool _gdsl_bintree_is_root (const _gdsl_bintree_t *T*)

Check if a low-level binary tree is a root.

Note:
Complexity: O(1)

Precondition:
T must be a non-empty _gdsl_bintree_t.

Parameters:
T The low-level binary tree to check.

Returns:
TRUE if the low-level binary tree T is a root.
FALSE if the low-level binary tree T is not a root.

See also:
_gdsl_bintree_is_empty()(p. 14)
_gdsl_bintree_is_leaf()(p. 15)

3.1.2.16 _gdsl_bintree_t _gdsl_bintree_map_infix (const _gdsl_bintree_t *T*, const gdsl_bintree_map_func_t *MAP_F*, void * *USER_DATA*)

Parse a low-level binary tree in infixed order.

Parse all nodes of the low-level binary tree T in infixed order. The MAP_F function is called on each node with the USER_DATA argument. If MAP_F returns GDSL_MAP_STOP, then **_gdsl_bintree_map_infix()**(p. 16) stops and returns its last examinated node.

Note:
Complexity: O(|T|)

Precondition:
MAP_F != NULL

Parameters:
T The low-level binary tree to map.

MAP_F The map function.

USER_DATA User's datas.

Returns:
the first node for which MAP_F returns GDSL_MAP_STOP.
NULL when the parsing is done.

See also:
_gdsl_bintree_map_prefix()(p. 17)
_gdsl_bintree_map_postfix()(p. 16)

3.1.2.17 _gdsl_bintree_t _gdsl_bintree_map_postfix (const _gdsl_bintree_t *T*, const gdsl_bintree_map_func_t *MAP_F*, void * *USER_DATA*)

Parse a low-level binary tree in postfixed order.

Parse all nodes of the low-level binary tree T in postfixed order. The MAP_F function is called on each node with the USER_DATA argument. If MAP_F returns GDSL_MAP_STOP, then **_gdsl_bintree_map_postfix()**(p. 16) stops and returns its last examinated node.

Note:
Complexity: O(|T|)

Precondition:
MAP_F != NULL

Parameters:
T The low-level binary tree to map.

MAP_F The map function.

USER_DATA User's datas.

Returns:
the first node for which MAP_F returns GDSL_MAP_STOP.
NULL when the parsing is done.

See also:
_gdsl_bintree_map_prefix()(p. 17)
_gdsl_bintree_map_infix()(p. 16)

3.1.2.18 _gdsl_bintree_t _gdsl_bintree_map_prefix (const _gdsl_bintree_t *T*, const gdsl_bintree_map_func_t *MAP_F*, void * *USER_DATA*)

Parse a low-level binary tree in prefixed order.

Parse all nodes of the low-level binary tree T in prefixed order. The MAP_F function is called on each node with the USER_DATA argument. If MAP_F returns GDSL_MAP_STOP, then **_gdsl_bintree_map_prefix()**(p. 17) stops and returns its last examinated node.

Note:
Complexity: O(|T|)

Precondition:
MAP_F != NULL

Parameters:
T The low-level binary tree to map.

MAP_F The map function.

USER_DATA User's datas.

Returns:
the first node for which MAP_F returns GDSL_MAP_STOP.
NULL when the parsing is done.

See also:
_gdsl_bintree_map_infix()(p. 16)
_gdsl_bintree_map_postfix()(p. 16)

3.1.2.19 _gdsl_bintree_t _gdsl_bintree_rotate_left (_gdsl_bintree_t * *T*)

Left rotate a low-level binary tree.

Do a left rotation of the low-level binary tree T.

Note:
Complexity: O(1)

Precondition:
T & r(T) must be non-empty _gdsl_bintree_t.

Parameters:
T The low-level binary tree to rotate.

Returns:
the modified T left-rotated.

See also:
_gdsl_bintree_rotate_right()(p. 18)
_gdsl_bintree_rotate_left_right()(p. 18)
_gdsl_bintree_rotate_right_left()(p. 19)

3.1.2.20 _gdsl_bintree_t _gdsl_bintree_rotate_left_right (_gdsl_bintree_t * *T*)

Left-right rotate a low-level binary tree.

Do a double left-right rotation of the low-level binary tree T.

Note:
Complexity: O(1)

Precondition:
T & l(T) & r(l(T)) must be non-empty _gdsl_bintree_t.

Parameters:
T The low-level binary tree to rotate.

Returns:
the modified T left-right-rotated.

See also:
_gdsl_bintree_rotate_left()(p. 17)
_gdsl_bintree_rotate_right()(p. 18)
_gdsl_bintree_rotate_right_left()(p. 19)

3.1.2.21 _gdsl_bintree_t _gdsl_bintree_rotate_right (_gdsl_bintree_t * *T*)

Right rotate a low-level binary tree.

Do a right rotation of the low-level binary tree T.

Note:
Complexity: O(1)

Precondition:
T & l(T) must be non-empty _gdsl_bintree_t.

Parameters:
T The low-level binary tree to rotate.

Returns:
the modified T right-rotated.

See also:
_gdsl_bintree_rotate_left()(p. 17)
_gdsl_bintree_rotate_left_right()(p. 18)
_gdsl_bintree_rotate_right_left()(p. 19)

3.1.2.22 _gdsl_bintree_t _gdsl_bintree_rotate_right_left (_gdsl_bintree_t * *T*)

Right-left rotate a low-level binary tree.

Do a double right-left rotation of the low-level binary tree T.

Note:
Complexity: O(1)

Precondition:
T & r(T) & l(r(T)) must be non-empty _gdsl_bintree_t.

Parameters:
T The low-level binary tree to rotate.

Returns:
the modified T right-left-rotated.

See also:
_gdsl_bintree_rotate_left()(p. 17)
_gdsl_bintree_rotate_right()(p. 18)
_gdsl_bintree_rotate_left_right()(p. 18)

3.1.2.23 void _gdsl_bintree_set_content (_gdsl_bintree_t *T*, const gdsl_element_t *E*)

Set the root element of a low-level binary tree.

Modify the root element of the low-level binary tree T to E.

Note:
Complexity: O(1)

Precondition:
T must be a non-empty _gdsl_bintree_t.

Parameters:
T The low-level binary tree to modify.

E The new T's root content.

See also:
_gdsl_bintree_get_content(p. 11)

3.1.2.24 void _gdsl_bintree_set_left (_gdsl_bintree_t *T*, const _gdsl_bintree_t *L*)

Set left sub-tree of a low-level binary tree.

Modify the left sub-tree of the low-level binary tree T to L.

Note:
Complexity: O(1)

Precondition:
T must be a non-empty _gdsl_bintree_t.

Parameters:
T The low-level binary tree to modify.

L The new T's left sub-tree.

See also:
_gdsl_bintree_set_right()(p. 21)
_gdsl_bintree_get_left()(p. 11)
_gdsl_bintree_get_right()(p. 13)

3.1.2.25 void _gdsl_bintree_set_parent (_gdsl_bintree_t *T*, const _gdsl_bintree_t *P*)

Set the parent tree of a low-level binary tree.

Modify the parent of the low-level binary tree T to P.

Note:
Complexity: O(1)

Precondition:
T must be a non-empty _gdsl_bintree_t.

Parameters:

T The low-level binary tree to modify.

P The new T's parent.

See also:

_gdsl_bintree_get_parent()(p. 12)

3.1.2.26 void _gdsl_bintree_set_right (_gdsl_bintree_t *T*, const _gdsl_bintree_t *R*)

Set right sub-tree of a low-level binary tree.

Modify the right sub-tree of the low-level binary tree T to R.

Note:

Complexity: O(1)

Precondition:

T must be a non-empty _gdsl_bintree_t.

Parameters:

T The low-level binary tree to modify.

R The new T's right sub-tree.

See also:

_gdsl_bintree_set_left()(p. 20)
_gdsl_bintree_get_left()(p. 11)
_gdsl_bintree_get_right()(p. 13)

3.1.2.27 void _gdsl_bintree_write (const _gdsl_bintree_t *T*, const gdsl_write_func_t *WRITE_F*, FILE * *OUTPUT_FILE*, void * *USER_DATA*)

Write the content of all nodes of a low-level binary tree to a file.

Write the nodes contents of the low-level binary tree T to OUTPUT_FILE, using WRITE_F function. Additionnal USER_DATA argument could be passed to WRITE_F.

Note:

Complexity: O(|T|)

Precondition:

WRITE_F != NULL & OUTPUT_FILE != NULL

Parameters:

T The low-level binary tree to write.

WRITE_F The write function.

OUTPUT_FILE The file where to write T's nodes.

USER_DATA User's datas passed to WRITE_F.

See also:

_gdsl_bintree_write_xml()(p. 22)
_gdsl_bintree_dump()(p. 9)

3.1.2.28 void _gdsl_bintree_write_xml (const _gdsl_bintree_t *T*, const gdsl_write_func_t *WRITE_F*, FILE * *OUTPUT_FILE*, void * *USER_DATA*)

Write the content of a low-level binary tree to a file into XML.

Write the nodes contents of the low-level binary tree T to OUTPUT_FILE, into XML language. If WRITE_F != NULL, then uses WRITE_F function to write T's nodes content to OUTPUT_FILE. Additionnal USER_DATA argument could be passed to WRITE_F.

Note:

Complexity: O(|T|)

Precondition:

OUTPUT_FILE != NULL

Parameters:

T The low-level binary tree to write.

WRITE_F The write function.

OUTPUT_FILE The file where to write T's nodes.

USER_DATA User's datas passed to WRITE_F.

See also:

_gdsl_bintree_write()(p. 21)
_gdsl_bintree_dump()(p. 9)

3.2 Low-level binary search tree manipulation module

Typedefs

- typedef **_gdsl_bintree_t _gdsl_bstree_t**

  GDSL low-level binary search tree type.

- typedef int(* **gdsl_bstree_map_func_t**)(**_gdsl_bstree_t** TREE, void *USER_DATA)

  GDSL low-level binary search tree map function type.

Functions

- **_gdsl_bstree_t _gdsl_bstree_alloc** (const **gdsl_element_t** E)

  Create a new low-level binary search tree.

- void **_gdsl_bstree_free** (**_gdsl_bstree_t** T, const **gdsl_free_func_t** FREE_F)

  Destroy a low-level binary search tree.

- **_gdsl_bstree_t _gdsl_bstree_copy** (const **_gdsl_bstree_t** T, const **gdsl_copy_func_t** COPY_F)

  Copy a low-level binary search tree.

- **bool _gdsl_bstree_is_empty** (const **_gdsl_bstree_t** T)

  Check if a low-level binary search tree is empty.

- **bool _gdsl_bstree_is_leaf** (const **_gdsl_bstree_t** T)

  Check if a low-level binary search tree is reduced to a leaf.

- **gdsl_element_t _gdsl_bstree_get_content** (const **_gdsl_bstree_t** T)

  Get the root content of a low-level binary search tree.

- **bool _gdsl_bstree_is_root** (const **_gdsl_bstree_t** T)

  Check if a low-level binary search tree is a root.

- **_gdsl_bstree_t _gdsl_bstree_get_parent** (const **_gdsl_bstree_t** T)

  Get the parent tree of a low-level binary search tree.

- **_gdsl_bstree_t _gdsl_bstree_get_left** (const **_gdsl_bstree_t** T)

  Get the left sub-tree of a low-level binary search tree.

- **_gdsl_bstree_t _gdsl_bstree_get_right** (const **_gdsl_bstree_t** T)

  Get the right sub-tree of a low-level binary search tree.

- **ulong _gdsl_bstree_get_size** (const **_gdsl_bstree_t** T)

  Get the size of a low-level binary search tree.

- **ulong _gdsl_bstree_get_height** (const **_gdsl_bstree_t** T)

  Get the height of a low-level binary search tree.

- **_gdsl_bstree_t _gdsl_bstree_insert** (**_gdsl_bstree_t** *T, const **gdsl_compare_func_t** COMP_F, const **gdsl_element_t** VALUE, int *RESULT)

  Insert an element into a low-level binary search tree if it's not found or return it.

- **gdsl_element_t _gdsl_bstree_remove** (**_gdsl_bstree_t** *T, const **gdsl_compare_func_t** COMP_F, const **gdsl_element_t** VALUE)

  Remove an element from a low-level binary search tree.

- **_gdsl_bstree_t _gdsl_bstree_search** (const **_gdsl_bstree_t** T, const **gdsl_compare_func_t** COMP_F, const **gdsl_element_t** VALUE)

  Search for a particular element into a low-level binary search tree.

- **_gdsl_bstree_t _gdsl_bstree_search_next** (const **_gdsl_bstree_t** T, const **gdsl_compare_func_t** COMP_F, const **gdsl_element_t** VALUE)

  Search for the next element of a particular element into a low-level binary search tree, according to the binary search tree order.

- **_gdsl_bstree_t _gdsl_bstree_map_prefix** (const **_gdsl_bstree_t** T, const **gdsl_bstree_map_func_t** MAP_F, void *USER_DATA)

  Parse a low-level binary search tree in prefixed order.

- **_gdsl_bstree_t _gdsl_bstree_map_infix** (const **_gdsl_bstree_t** T, const **gdsl_bstree_map_func_t** MAP_F, void *USER_DATA)

  Parse a low-level binary search tree in infixed order.

- **_gdsl_bstree_t _gdsl_bstree_map_postfix** (const **_gdsl_bstree_t** T, const **gdsl_bstree_map_func_t** MAP_F, void *USER_DATA)

  Parse a low-level binary search tree in postfixed order.

- void **_gdsl_bstree_write** (const **_gdsl_bstree_t** T, const **gdsl_write_func_t** WRITE_F, FILE *OUTPUT_FILE, void *USER_DATA)

  Write the content of all nodes of a low-level binary search tree to a file.

- void **_gdsl_bstree_write_xml** (const **_gdsl_bstree_t** T, const **gdsl_write_func_t** WRITE_F, FILE *OUTPUT_FILE, void *USER_DATA)

  Write the content of a low-level binary search tree to a file into XML.

- void **_gdsl_bstree_dump** (const **_gdsl_bstree_t** T, const **gdsl_write_func_t** WRITE_F, FILE *OUTPUT_FILE, void *USER_DATA)

  Dump the internal structure of a low-level binary search tree to a file.

3.2.1 Typedef Documentation

3.2.1.1 typedef _gdsl_bintree_t _gdsl_bstree_t

GDSL low-level binary search tree type.

This type is voluntary opaque. Variables of this kind could'nt be directly used, but by the functions of this module.

Definition at line 51 of file _gdsl_bstree.h.

3.2.1.2 typedef int(* gdsl_bstree_map_func_t)(_gdsl_bstree_t TREE, void *USER_DATA)

GDSL low-level binary search tree map function type.

Parameters:

TREE The low-level binary search tree to be mapped

USER_DATA The user datas to pass to this function

Returns:

Must be 0 if the mapping must be stopped, != 0 otherwise.

Definition at line 59 of file _gdsl_bstree.h.

3.2.2 Function Documentation

3.2.2.1 _gdsl_bstree_t _gdsl_bstree_alloc (const gdsl_element_t *E*)

Create a new low-level binary search tree.

Allocate a new low-level binary search tree data structure. Its root content is sets to E and its left and right sons are set to NULL.

Note:
Complexity: O(1)

Precondition:
nothing.

Parameters:
E The root content of the new low-level binary search tree to create.

Returns:
the newly allocated low-level binary search tree in case of success.
NULL in case of insufficient memory.

See also:
_gdsl_bstree_free()(p. 27)

3.2.2.2 _gdsl_bstree_t _gdsl_bstree_copy (const _gdsl_bstree_t *T*, const gdsl_copy_func_t *COPY_F*)

Copy a low-level binary search tree.

Create and return a copy of the low-level binary search tree T using COPY_F on each T's element to copy them.

Note:
Complexity: O(|T|)

Precondition:
COPY_F != NULL

Parameters:
T The low-level binary search tree to copy.
COPY_F The function used to copy T's nodes contents.

Returns:
a copy of T in case of success.
NULL if _gdsl_bstree_is_empty (T) == TRUE or in case of insufficient memory.

See also:
_gdsl_bstree_alloc()(p. 25)
_gdsl_bstree_free()(p. 27)
_gdsl_bstree_is_empty()(p. 31)

3.2.2.3 void _gdsl_bstree_dump (const _gdsl_bstree_t *T*, const gdsl_write_func_t *WRITE_F*, FILE * *OUTPUT_FILE*, void * *USER_DATA*)

Dump the internal structure of a low-level binary search tree to a file.

Dump the structure of the low-level binary search tree T to OUTPUT_FILE. If WRITE_F != NULL, then use WRITE_F function to write T's nodes content to OUTPUT_FILE. Additionnal USER_DATA argument could be passed to WRITE_F.

Note:
Complexity: O(|T|)

Precondition:
OUTPUT_FILE != NULL

Parameters:
T The low-level binary search tree to dump.
WRITE_F The write function.
OUTPUT_FILE The file where to write T's nodes.
USER_DATA User's datas passed to WRITE_F.

See also:
_gdsl_bstree_write()(p. 36)
_gdsl_bstree_write_xml()(p. 37)

3.2.2.4 void _gdsl_bstree_free (_gdsl_bstree_t *T*, const gdsl_free_func_t *FREE_F*)

Destroy a low-level binary search tree.

Flush and destroy the low-level binary search tree T. If FREE_F != NULL, FREE_F function is used to deallocate each T's element. Otherwise nothing is done with T's elements.

Note:
Complexity: O(|T|)

Precondition:
nothing.

Parameters:
T The low-level binary search tree to destroy.
FREE_F The function used to deallocate T's nodes contents.

See also:
_gdsl_bstree_alloc()(p. 25)

3.2.2.5 gdsl_element_t _gdsl_bstree_get_content (const _gdsl_bstree_t *T*)

Get the root content of a low-level binary search tree.

Note:
Complexity: O(1)

Precondition:
T must be a non-empty _gdsl_bstree_t.

Parameters:
T The low-level binary search tree to use.

Returns:
the root's content of the low-level binary search tree T.

3.2.2.6 ulong _gdsl_bstree_get_height (const _gdsl_bstree_t *T*)

Get the height of a low-level binary search tree.

Compute the height of the low-level binary search tree T (noted h(T)).

Note:
Complexity: O(|T|)

Precondition:
nothing.

Parameters:
T The low-level binary search tree to compute the height from.

Returns:
the height of T.

See also:
_gdsl_bstree_get_size()(p. 30)

3.2.2.7 _gdsl_bstree_t _gdsl_bstree_get_left (const _gdsl_bstree_t *T*)

Get the left sub-tree of a low-level binary search tree.

Note:
Complexity: O(1)

Precondition:
T must be a non-empty _gdsl_bstree_t.

Parameters:
T The low-level binary search tree to use.

Returns:
the left sub-tree of the low-level binary search tree T if T has a left sub-tree.
NULL if the low-level binary search tree T has no left sub-tree.

See also:
_gdsl_bstree_get_right()(p. 29)

3.2.2.8 _gdsl_bstree_t _gdsl_bstree_get_parent (const _gdsl_bstree_t *T*)

Get the parent tree of a low-level binary search tree.

Note:
Complexity: O(1)

Precondition:
T must be a non-empty _gdsl_bstree_t.

Parameters:
T The low-level binary search tree to use.

Returns:
the parent of the low-level binary search tree T if T isn't a root.
NULL if the low-level binary search tree T is a root (ie. T has no parent).

See also:
_gdsl_bstree_is_root()(p. 32)

3.2.2.9 _gdsl_bstree_t _gdsl_bstree_get_right (const _gdsl_bstree_t *T*)

Get the right sub-tree of a low-level binary search tree.

Note:
Complexity: O(1)

Precondition:
T must be a non-empty _gdsl_bstree_t.

Parameters:
T The low-level binary search tree to use.

Returns:
the right sub-tree of the low-level binary search tree T if T has a right sub-tree.
NULL if the low-level binary search tree T has no right sub-tree.

See also:
_gdsl_bstree_get_left()(p. 28)

3.2.2.10 ulong _gdsl_bstree_get_size (const _gdsl_bstree_t *T*)

Get the size of a low-level binary search tree.

Note:
Complexity: O(|T|)

Precondition:
nothing.

Parameters:
T The low-level binary search tree to compute the size from.

Returns:
the number of elements of T (noted |T|).

See also:
_gdsl_bstree_get_height()(p. 28)

3.2.2.11 _gdsl_bstree_t _gdsl_bstree_insert (_gdsl_bstree_t * *T*, const gdsl_compare_func_t *COMP_F*, const gdsl_element_t *VALUE*, int * *RESULT*)

Insert an element into a low-level binary search tree if it's not found or return it.

Search for the first element E equal to VALUE into the low-level binary search tree T, by using COMP_F function to find it. If an element E equal to VALUE is found, then it's returned. If no element equal to VALUE is found, then E is inserted and its root returned.

Note:
Complexity: O(h(T)), where log2(|T|) <= h(T) <= |T|-1

Precondition:
COMP_F != NULL & RESULT != NULL

Parameters:
T The reference of the low-level binary search tree to use.

COMP_F The comparison function to use to compare T's elements with VALUE to find E.

VALUE The value used to search for the element E.

RESULT The address where the result code will be stored.

Returns:

the root containing E and RESULT = GDSL_INSERTED if E is inserted.
the root containing E and RESULT = GDSL_ERR_DUPLICATE_ENTRY if E is not inserted.
NULL and RESULT = GDSL_ERR_MEM_ALLOC in case of failure.

See also:

_gdsl_bstree_search()(p. 35)
_gdsl_bstree_remove()(p. 34)

3.2.2.12 bool _gdsl_bstree_is_empty (const _gdsl_bstree_t *T*)

Check if a low-level binary search tree is empty.

Note:

Complexity: O(1)

Precondition:

nothing.

Parameters:

T The low-level binary search tree to check.

Returns:

TRUE if the low-level binary search tree T is empty.
FALSE if the low-level binary search tree T is not empty.

See also:

_gdsl_bstree_is_leaf()(p. 31)
_gdsl_bstree_is_root()(p. 32)

3.2.2.13 bool _gdsl_bstree_is_leaf (const _gdsl_bstree_t *T*)

Check if a low-level binary search tree is reduced to a leaf.

Note:

Complexity: O(1)

Precondition:

T must be a non-empty _gdsl_bstree_t.

Parameters:
T The low-level binary search tree to check.

Returns:
TRUE if the low-level binary search tree T is a leaf.
FALSE if the low-level binary search tree T is not a leaf.

See also:
_gdsl_bstree_is_empty()(p. 31)
_gdsl_bstree_is_root()(p. 32)

3.2.2.14 bool _gdsl_bstree_is_root (const _gdsl_bstree_t *T*)

Check if a low-level binary search tree is a root.

Note:
Complexity: O(1)

Precondition:
T must be a non-empty _gdsl_bstree_t.

Parameters:
T The low-level binary search tree to check.

Returns:
TRUE if the low-level binary search tree T is a root.
FALSE if the low-level binary search tree T is not a root.

See also:
_gdsl_bstree_is_empty()(p. 31)
_gdsl_bstree_is_leaf()(p. 31)

3.2.2.15 _gdsl_bstree_t _gdsl_bstree_map_infix (const _gdsl_bstree_t *T*, const gdsl_bstree_map_func_t *MAP_F*, void * *USER_DATA*)

Parse a low-level binary search tree in infixed order.

Parse all nodes of the low-level binary search tree T in infixed order. The MAP_F function is called on each node with the USER_DATA argument. If MAP_F returns GDSL_MAP_STOP, then **_gdsl_bstree_map_infix()**(p. 32) stops and returns its last examinated node.

Note:
Complexity: O(|T|)

Precondition:
MAP_F != NULL

Parameters:

T The low-level binary search tree to map.

MAP_F The map function.

USER_DATA User's datas passed to MAP_F.

Returns:

the first node for which MAP_F returns GDSL_MAP_STOP.
NULL when the parsing is done.

See also:

_gdsl_bstree_map_prefix()(p. 34)
_gdsl_bstree_map_postfix()(p. 33)

3.2.2.16 _gdsl_bstree_t _gdsl_bstree_map_postfix (const _gdsl_bstree_t *T*, const gdsl_bstree_map_func_t *MAP_F*, void * *USER_DATA*)

Parse a low-level binary search tree in postfixed order.

Parse all nodes of the low-level binary search tree T in postfixed order. The MAP_F function is called on each node with the USER_DATA argument. If MAP_F returns GDSL_MAP_STOP, then **_gdsl_bstree_map_postfix()**(p. 33) stops and returns its last examinated node.

Note:

Complexity: O(|T|)

Precondition:

MAP_F != NULL

Parameters:

T The low-level binary search tree to map.

MAP_F The map function.

USER_DATA User's datas passed to MAP_F.

Returns:

the first node for which MAP_F returns GDSL_MAP_STOP.
NULL when the parsing is done.

See also:

_gdsl_bstree_map_prefix()(p. 34)
_gdsl_bstree_map_infix()(p. 32)

3.2.2.17 _gdsl_bstree_t _gdsl_bstree_map_prefix (const _gdsl_bstree_t *T*, const gdsl_bstree_map_func_t *MAP_F*, void * *USER_DATA*)

Parse a low-level binary search tree in prefixed order.

Parse all nodes of the low-level binary search tree T in prefixed order. The MAP_F function is called on each node with the USER_DATA argument. If MAP_F returns GDSL_MAP_STOP, then **_gdsl_bstree_map_prefix()**(p. 34) stops and returns its last examinated node.

Note:
Complexity: O(|T|)

Precondition:
MAP_F != NULL

Parameters:
T The low-level binary search tree to map.

MAP_F The map function.

USER_DATA User's datas passed to MAP_F.

Returns:
the first node for which MAP_F returns GDSL_MAP_STOP.
NULL when the parsing is done.

See also:
_gdsl_bstree_map_infix()(p. 32)
_gdsl_bstree_map_postfix()(p. 33)

3.2.2.18 gdsl_element_t _gdsl_bstree_remove (_gdsl_bstree_t * *T*, const gdsl_compare_func_t *COMP_F*, const gdsl_element_t *VALUE*)

Remove an element from a low-level binary search tree.

Remove from the low-level binary search tree T the first founded element E equal to VALUE, by using COMP_F function to compare T's elements. If E is found, it is removed from T.

Note:
Complexity: O(h(T)), where log2(|T|) <= h(T) <= |T|-1
The resulting T is modified by examinating the left sub-tree from the founded e.

Precondition:
COMP_F != NULL

Parameters:

T The reference of the low-level binary search tree to modify.

COMP_F The comparison function to use to compare T's elements with VALUE to find the element e to remove.

VALUE The value that must be used by COMP_F to find the element e to remove.

Returns:

the fisrt founded element equal to VALUE in T.
NULL if no element equal to VALUE is found or if T is empty.

See also:

_gdsl_bstree_insert()(p. 30)
_gdsl_bstree_search()(p. 35)

3.2.2.19 _gdsl_bstree_t _gdsl_bstree_search (const _gdsl_bstree_t *T*, const gdsl_compare_func_t *COMP_F*, const gdsl_element_t *VALUE*)

Search for a particular element into a low-level binary search tree.

Search the first element E equal to VALUE in the low-level binary search tree T, by using COMP_F function to find it.

Note:

Complexity: O(h(T)), where log2(|T|) <= h(T) <= |T|-1

Precondition:

COMP_F != NULL

Parameters:

T The low-level binary search tree to use.

COMP_F The comparison function to use to compare T's elements with VALUE to find the element E.

VALUE The value that must be used by COMP_F to find the element E.

Returns:

the root of the tree containing E if it's found.
NULL if VALUE is not found in T.

See also:

_gdsl_bstree_insert()(p. 30)
_gdsl_bstree_remove()(p. 34)

3.2.2.20 _gdsl_bstree_t _gdsl_bstree_search_next (const _gdsl_bstree_t *T*, const gdsl_compare_func_t *COMP_F*, const gdsl_element_t *VALUE*)

Search for the next element of a particular element into a low-level binary search tree, according to the binary search tree order.

Search for an element E in the low-level binary search tree T, by using COMP_F function to find the first element E equal to VALUE.

Note:
Complexity: O(h(T)), where log2(|T|) <= h(T) <= |T|-1

Precondition:
COMP_F != NULL

Parameters:

T The low-level binary search tree to use.

COMP_F The comparison function to use to compare T's elements with VALUE to find the element E.

VALUE The value that must be used by COMP_F to find the element E.

Returns:
the root of the tree containing the successor of E if it's found.
NULL if VALUE is not found in T or if E has no sucessor.

3.2.2.21 void _gdsl_bstree_write (const _gdsl_bstree_t *T*, const gdsl_write_func_t *WRITE_F*, FILE * *OUTPUT_FILE*, void * *USER_DATA*)

Write the content of all nodes of a low-level binary search tree to a file.

Write the nodes contents of the low-level binary search tree T to OUTPUT_FILE, using WRITE_F function. Additionnal USER_DATA argument could be passed to WRITE_F.

Note:
Complexity: O(|T|)

Precondition:
WRITE_F != NULL& OUTPUT_FILE != NULL

Parameters:

T The low-level binary search tree to write.

WRITE_F The write function.

OUTPUT_FILE The file where to write T's nodes.

USER_DATA User's datas passed to WRITE_F.

See also:
_gdsl_bstree_write_xml()(p. 37)
_gdsl_bstree_dump()(p. 27)

3.2.2.22 void _gdsl_bstree_write_xml (const _gdsl_bstree_t *T*, const gdsl_write_func_t *WRITE_F*, FILE * *OUTPUT_FILE*, void * *USER_DATA*)

Write the content of a low-level binary search tree to a file into XML.

Write the nodes contents of the low-level binary search tree T to OUTPUT_FILE, into XML language. If WRITE_F != NULL, then use WRITE_F function to write T's nodes contents to OUTPUT_FILE. Additionnal USER_DATA argument could be passed to WRITE_F.

Note:
Complexity: O(|T|)

Precondition:
OUTPUT_FILE != NULL

Parameters:
T The low-level binary search tree to write.

WRITE_F The write function.

OUTPUT_FILE The file where to write T's nodes.

USER_DATA User's datas passed to WRITE_F.

See also:
_gdsl_bstree_write()(p. 36)
_gdsl_bstree_dump()(p. 27)

3.3 Low-level doubly-linked list manipulation module

Typedefs

- typedef **_gdsl_node_t _gdsl_list_t**

  GDSL low-level doubly-linked list type.

- typedef int(* **_gdsl_list_map_func_t**)(**_gdsl_node_t** NODE, void *USER_DATA)

  GDSL low-level doubly-linked list map function type.

Functions

- **_gdsl_list_t _gdsl_list_alloc** (const **gdsl_element_t** E)

  Create a new low-level list.

- void **_gdsl_list_free** (**_gdsl_list_t** L, const **gdsl_free_func_t** FREE_F)

  Destroy a low-level list.

- **bool _gdsl_list_is_empty** (const **_gdsl_list_t** L)

  Check if a low-level list is empty.

- **ulong _gdsl_list_get_size** (const **_gdsl_list_t** L)

  Get the size of a low-level list.

- void **_gdsl_list_link** (**_gdsl_list_t** L1, **_gdsl_list_t** L2)

  Link two low-level lists together.

- void **_gdsl_list_insert_after** (**_gdsl_list_t** L, **_gdsl_list_t** PREV)

  Insert a low-level list after another one.

- void **_gdsl_list_insert_before** (**_gdsl_list_t** L, **_gdsl_list_t** SUCC)

  Insert a low-level list before another one.

- void **_gdsl_list_remove** (**_gdsl_node_t** NODE)

  Remove a node from a low-level list.

- **_gdsl_list_t _gdsl_list_search** (**_gdsl_list_t** L, const **gdsl_compare_func_t** COMP_F, void *VALUE)

  Search for a particular node in a low-level list.

- **_gdsl_list_t _gdsl_list_map_forward** (const **_gdsl_list_t** L, const **_gdsl_list_map_func_t** MAP_F, void *USER_DATA)

  Parse a low-level list in forward order.

- **_gdsl_list_t _gdsl_list_map_backward** (const **_gdsl_list_t** L, const **_gdsl_list_map_func_t** MAP_F, void *USER_DATA)

  Parse a low-level list in backward order.

- void **_gdsl_list_write** (const **_gdsl_list_t** L, const **gdsl_write_func_t** WRITE_F, FILE *OUTPUT_FILE, void *USER_DATA)

  Write the contents of all nodes of a low-level list to a file.

- void **_gdsl_list_write_xml** (const **_gdsl_list_t** L, const **gdsl_write_func_t** WRITE_F, FILE *OUTPUT_FILE, void *USER_DATA)

  Write the contents of all nodes of a low-level list to a file into XML.

- void **_gdsl_list_dump** (const **_gdsl_list_t** L, const **gdsl_write_func_t** WRITE_F, FILE *OUTPUT_FILE, void *USER_DATA)

  Dump the internal structure of a low-level list to a file.

3.3.1 Typedef Documentation

3.3.1.1 typedef int(* _gdsl_list_map_func_t)(_gdsl_node_t NODE, void* USER_DATA)

GDSL low-level doubly-linked list map function type.

Parameters:

NODE The low-level node to map.

USER_DATA The user datas to pass to this function.

Returns:

0 if the mapping must be stopped, another value otherwise.

Definition at line 61 of file _gdsl_list.h.

3.3.1.2 typedef _gdsl_node_t _gdsl_list_t

GDSL low-level doubly-linked list type.

This type is voluntary opaque. Variables of this kind could'nt be directly used, but by the functions of this module.

Definition at line 53 of file _gdsl_list.h.

3.3.2 Function Documentation

3.3.2.1 _gdsl_list_t _gdsl_list_alloc (const gdsl_element_t *E*)

Create a new low-level list.

Allocate a new low-level list data structure which have only one node. The node's content is set to E.

Note:
Complexity: O(1)

Precondition:
nothing.

Parameters:
E The content of the first node of the new low-level list to create

Returns:
the newly allocated low-level list in case of success.
NULL in case of insufficient memory.

See also:
_gdsl_list_free()(p. 41)

3.3.2.2 void _gdsl_list_dump (const _gdsl_list_t *L*, const gdsl_write_func_t *WRITE_F*, FILE * *OUTPUT_FILE*, void * *USER_DATA*)

Dump the internal structure of a low-level list to a file.

Dump the structure of the low-level list L to OUTPUT_FILE. If WRITE_F != NULL, then uses WRITE_F function to write L's nodes contents to OUTPUT_FILE. Additionnal USER_DATA argument could be passed to WRITE_F.

Note:
Complexity: O(|L|)

Precondition:
OUTPUT_FILE != NULL

Parameters:
L The low-level list to dump
WRITE_F The write function
OUTPUT_FILE The file where to write L's nodes
USER_DATA User's datas passed to WRITE_F

See also:
_gdsl_list_write()(p. 45)
_gdsl_list_write_xml()(p. 46)

3.3.2.3 void _gdsl_list_free (_gdsl_list_t *L*, const gdsl_free_func_t *FREE_F*)

Destroy a low-level list.

Flush and destroy the low-level list L. If FREE_F != NULL, then the FREE_F function is used to deallocated each L's element. Otherwise, nothing is done with L's elements.

Note:
Complexity: O(|L|)

Precondition:
nothing.

Parameters:
L The low-level list to destroy

FREE_F The function used to deallocated L's nodes contents

See also:
_gdsl_list_alloc()(p. 40)

3.3.2.4 ulong _gdsl_list_get_size (const _gdsl_list_t *L*)

Get the size of a low-level list.

Note:
Complexity: O(|L|)

Precondition:
nothing

Parameters:
L The low-level list to use

Returns:
the number of elements of L (noted |L|).

3.3.2.5 void _gdsl_list_insert_after (_gdsl_list_t *L*, _gdsl_list_t *PREV*)

Insert a low-level list after another one.

Insert the low-level list L after the low-level list PREV.

Note:
Complexity: O(|L|)

Precondition:
L & PREV must be non-empty _gdsl_list_t

Parameters:
L The low-level list to link after PREV

PREV The low-level list that will be linked before L

See also:
_gdsl_list_insert_before()(p. 42)
_gdsl_list_remove()(p. 44)

3.3.2.6 void _gdsl_list_insert_before (_gdsl_list_t *L*, _gdsl_list_t *SUCC*)

Insert a low-level list before another one.

Insert the low-level list L before the low-level list SUCC.

Note:
Complexity: O(|L|)

Precondition:
L & SUCC must be non-empty _gdsl_list_t

Parameters:
L The low-level list to link before SUCC

SUCC The low-level list that will be linked after L

See also:
_gdsl_list_insert_after()(p. 41)
_gdsl_list_remove()(p. 44)

3.3.2.7 bool _gdsl_list_is_empty (const _gdsl_list_t *L*)

Check if a low-level list is empty.

Note:
Complexity: O(1)

Precondition:
nothing

Parameters:
L The low-level list to check

Returns:
TRUE if the low-level list L is empty.
FALSE if the low-level list L is not empty.

3.3.2.8 void _gdsl_list_link (_gdsl_list_t *L1*, _gdsl_list_t *L2*)

Link two low-level lists together.

Link the low-level list L2 after the end of the low-level list L1. So L1 is before L2.

Note:
Complexity: O(|L1|)

Precondition:
L1 & L2 must be non-empty _gdsl_list_t

Parameters:
L1 The low-level list to link before L2

L2 The low-level list to link after L1

3.3.2.9 _gdsl_list_t _gdsl_list_map_backward (const _gdsl_list_t *L*, const _gdsl_list_map_func_t *MAP_F*, void * *USER_DATA*)

Parse a low-level list in backward order.

Parse all nodes of the low-level list L in backward order. The MAP_F function is called on each node with the USER_DATA argument. If MAP_F returns GDSL_MAP_STOP, then **_gdsl_list_map_backward()**(p. 43) stops and returns its last examinated node.

Note:
Complexity: O(2 |L|)

Precondition:
L must be a non-empty _gdsl_list_t & MAP_F != NULL

Parameters:
L Th low-level list to map

MAP_F The map function

USER_DATA User's datas

Returns:
the first node for which MAP_F returns GDSL_MAP_STOP.
NULL when the parsing is done.

See also:
_gdsl_list_map_forward()(p. 44)

3.3.2.10 _gdsl_list_t _gdsl_list_map_forward (const _gdsl_list_t *L*, const _gdsl_list_map_func_t *MAP_F*, void * *USER_DATA*)

Parse a low-level list in forward order.

Parse all nodes of the low-level list L in forward order. The MAP_F function is called on each node with the USER_DATA argument. If MAP_F returns GDSL_MAP_STOP, then **_gdsl_list_map_forward()**(p. 44) stops and returns its last examinated node.

Note:
Complexity: O(|L|)

Precondition:
MAP_F != NULL

Parameters:
L Th low-level list to map

MAP_F The map function

USER_DATA User's datas

Returns:
the first node for which MAP_F returns GDSL_MAP_STOP.
NULL when the parsing is done.

See also:
_gdsl_list_map_backward()(p. 43)

3.3.2.11 void _gdsl_list_remove (_gdsl_node_t *NODE*)

Remove a node from a low-level list.

Unlink the node NODE from the low-level list in which it is inserted.

Note:
Complexity: O(1)

Precondition:
NODE must be a non-empty _gdsl_node_t

Parameters:
NODE The low-level node to unlink from the low-level list in which it's linked

See also:
_gdsl_list_insert_after()(p. 41)
_gdsl_list_insert_before()(p. 42)

3.3.2.12 _gdsl_list_t _gdsl_list_search (_gdsl_list_t *L*, const gdsl_compare_func_t *COMP_F*, void * *VALUE*)

Search for a particular node in a low-level list.

Research an element e in the low-level list L, by using COMP_F function to find the first element e equal to VALUE.

Note:
Complexity: O(|L|)

Precondition:
COMP_F != NULL

Parameters:
L The low-level list to use

COMP_F The comparison function to use to compare L's elements with VALUE to find the element e

VALUE The value that must be used by COMP_F to find the element e

Returns:
the sub-list starting by e if it's found.
NULL if VALUE is not found in L.

3.3.2.13 void _gdsl_list_write (const _gdsl_list_t *L*, const gdsl_write_func_t *WRITE_F*, FILE * *OUTPUT_FILE*, void * *USER_DATA*)

Write the contents of all nodes of a low-level list to a file.

Write the nodes contents of the low-level list L to OUTPUT_FILE, using WRITE_F function. Additionnal USER_DATA argument could be passed to WRITE_F.

Note:
Complexity: O(|L|)

Precondition:
WRITE_F != NULL & OUTPUT_FILE != NULL

Parameters:
L The low-level list to write

WRITE_F The write function

OUTPUT_FILE The file where to write L's nodes

USER_DATA User's datas passed to WRITE_F

See also:
_gdsl_list_write_xml()(p. 46)
_gdsl_list_dump()(p. 40)

3.3.2.14 void _gdsl_list_write_xml (const _gdsl_list_t *L*, const gdsl_write_func_t *WRITE_F*, FILE * *OUTPUT_FILE*, void * *USER_DATA*)

Write the contents of all nodes of a low-level list to a file into XML.

Write the nodes contents of the low-level list L to OUTPUT_FILE, into XML language. If WRITE_F != NULL, then uses WRITE_F function to write L's nodes contents to OUTPUT_FILE. Additionnal USER_DATA argument could be passed to WRITE_F.

Note:
Complexity: O(|L|)

Precondition:
OUTPUT_FILE != NULL

Parameters:
L The low-level list to write

WRITE_F The write function

OUTPUT_FILE The file where to write L's nodes

USER_DATA User's datas passed to WRITE_F

See also:
_gdsl_list_write()(p. 45)
_gdsl_list_dump()(p. 40)

3.4 Low-level node manipulation module

Typedefs

- typedef _gdsl_node * **_gdsl_node_t**

  GDSL low-level doubly linked node type.

Functions

- **_gdsl_node_t _gdsl_node_alloc** (void)

  Create a new low-level node.

- **gdsl_element_t _gdsl_node_free** (**_gdsl_node_t** NODE)

  Destroy a low-level node.

- **_gdsl_node_t _gdsl_node_get_succ** (const **_gdsl_node_t** NODE)

  Get the successor of a low-level node.

- **_gdsl_node_t _gdsl_node_get_pred** (const **_gdsl_node_t** NODE)

  Get the predecessor of a low-level node.

- **gdsl_element_t _gdsl_node_get_content** (const **_gdsl_node_t** NODE)

  Get the content of a low-level node.

- void **_gdsl_node_set_succ** (**_gdsl_node_t** NODE, const **_gdsl_node_t** SUCC)

  Set the successor of a low-level node.

- void **_gdsl_node_set_pred** (**_gdsl_node_t** NODE, const **_gdsl_node_t** PRED)

  Set the predecessor of a low-level node.

- void **_gdsl_node_set_content** (**_gdsl_node_t** NODE, const **gdsl_element_t** CONTENT)

  Set the content of a low-level node.

- void **_gdsl_node_link** (**_gdsl_node_t** NODE1, **_gdsl_node_t** NODE2)

  Link two low-level nodes together.

- void **_gdsl_node_unlink** (**_gdsl_node_t** NODE1, **_gdsl_node_t** NODE2)

Unlink two low-level nodes.

- void **_gdsl_node_write** (const **_gdsl_node_t** NODE, const **gdsl_write_func_t** WRITE_F, FILE *OUTPUT_FILE, void *USER_DATA)

  Write the content of a low-level node to a file.

- void **_gdsl_node_write_xml** (const **_gdsl_node_t** NODE, const **gdsl_write_func_t** WRITE_F, FILE *OUTPUT_FILE, void *USER_DATA)

  Write the content of a low-level node to a file into XML.

- void **_gdsl_node_dump** (const **_gdsl_node_t** NODE, const **gdsl_write_func_t** WRITE_F, FILE *OUTPUT_FILE, void *USER_DATA)

  Dump the internal structure of a low-level node to a file.

3.4.1 Typedef Documentation

3.4.1.1 typedef struct _gdsl_node* _gdsl_node_t

GDSL low-level doubly linked node type.

This type is voluntary opaque. Variables of this kind could'nt be directly used, but by the functions of this module.

Definition at line 49 of file _gdsl_node.h.

3.4.2 Function Documentation

3.4.2.1 _gdsl_node_t _gdsl_node_alloc (void)

Create a new low-level node.

Allocate a new low-level node data structure.

Note:
Complexity: O(1)

Precondition:
nothing.

Returns:
the newly allocated low-level node in case of success.
NULL in case of insufficient memory.

See also:
_gdsl_node_free()(p. 49)

3.4.2.2 void _gdsl_node_dump (const _gdsl_node_t *NODE*, const gdsl_write_func_t *WRITE_F*, FILE * *OUTPUT_FILE*, void * *USER_DATA*)

Dump the internal structure of a low-level node to a file.

Dump the structure of the low-level node NODE to OUTPUT_FILE. If WRITE_F != NULL, then uses WRITE_F function to write NODE's content to OUTPUT_FILE. Additionnal USER_DATA argument could be passed to WRITE_F.

Note:
Complexity: O(1)

Precondition:
NODE != NULL & OUTPUT_FILE != NULL

Parameters:
NODE The low-level node to dump.

WRITE_F The write function.

OUTPUT_FILE The file where to write NODE's content.

USER_DATA User's datas passed to WRITE_F.

See also:
_gdsl_node_write()(p. 53)
_gdsl_node_write_xml()(p. 54)

3.4.2.3 gdsl_element_t _gdsl_node_free (_gdsl_node_t *NODE*)

Destroy a low-level node.

Deallocate the low-level node NODE.

Note:
O(1)

Precondition:
NODE != NULL

Returns:
the content of NODE (without modification).

See also:
_gdsl_node_alloc()(p. 48)

3.4.2.4 gdsl_element_t _gdsl_node_get_content (const _gdsl_node_t *NODE*)

Get the content of a low-level node.

Note:
Complexity: O(1)

Precondition:
NODE != NULL

Parameters:
NODE The low-level node which we want to get the content from.

Returns:
the content of the low-level node NODE if NODE has a content.
NULL if the low-level node NODE has no content.

See also:
_gdsl_node_set_content()(p. 52)

3.4.2.5 _gdsl_node_t _gdsl_node_get_pred (const _gdsl_node_t *NODE*)

Get the predecessor of a low-level node.

Note:
Complexity: O(1)

Precondition:
NODE != NULL

Parameters:
NODE The low-level node which we want to get the predecessor from.

Returns:
the predecessor of the low-level node NODE if NODE has a predecessor.
NULL if the low-level node NODE has no predecessor.

See also:
_gdsl_node_get_succ()(p. 51)
_gdsl_node_set_succ()(p. 52)
_gdsl_node_set_pred()(p. 52)

3.4.2.6 _gdsl_node_t _gdsl_node_get_succ (const _gdsl_node_t *NODE*)

Get the successor of a low-level node.

Note:
Complexity: O(1)

Precondition:
NODE != NULL

Parameters:
NODE The low-level node which we want to get the successor from.

Returns:
the sucessor of the low-level node NODE if NODE has a successor.
NULL if the low-level node NODE has no successor.

See also:
_gdsl_node_get_pred()(p. 50)
_gdsl_node_set_succ()(p. 52)
_gdsl_node_set_pred()(p. 52)

3.4.2.7 void _gdsl_node_link (_gdsl_node_t *NODE1*, _gdsl_node_t *NODE2*)

Link two low-level nodes together.

Link the two low-level nodes NODE1 and NODE2 together. After the link, NODE1's successor is NODE2 and NODE2's predecessor is NODE1.

Note:
Complexity: O(1)

Precondition:
NODE1 != NULL & NODE2 != NULL

Parameters:
NODE1 The first low-level node to link to NODE2.

NODE2 The second low-level node to link from NODE1.

See also:
_gdsl_node_unlink()(p. 53)

3.4.2.8 void _gdsl_node_set_content (_gdsl_node_t *NODE*, const gdsl_element_t *CONTENT*)

Set the content of a low-level node.

Modifie the content of the low-level node NODE to CONTENT.

Note:
Complexity: O(1)

Precondition:
NODE != NULL

Parameters:
NODE The low-level node which want to change the content from.

CONTENT The new content of NODE.

See also:
_gdsl_node_get_content()(p. 50)

3.4.2.9 void _gdsl_node_set_pred (_gdsl_node_t *NODE*, const _gdsl_node_t *PRED*)

Set the predecessor of a low-level node.

Modifie the predecessor of the low-level node NODE to PRED.

Note:
Complexity: O(1)

Precondition:
NODE != NULL

Parameters:
NODE The low-level node which want to change the predecessor from.

PRED The new predecessor of NODE.

See also:
_gdsl_node_get_pred()(p. 50)

3.4.2.10 void _gdsl_node_set_succ (_gdsl_node_t *NODE*, const _gdsl_node_t *SUCC*)

Set the successor of a low-level node.

Modifie the sucessor of the low-level node NODE to SUCC.

Note:
Complexity: O(1)

Precondition:
NODE != NULL

Parameters:
NODE The low-level node which want to change the successor from.

SUCC The new successor of NODE.

See also:
_gdsl_node_get_succ()(p. 51)

3.4.2.11 void _gdsl_node_unlink (_gdsl_node_t *NODE1*, _gdsl_node_t *NODE2*)

Unlink two low-level nodes.

Unlink the two low-level nodes NODE1 and NODE2. After the unlink, NODE1's successor is NULL and NODE2's predecessor is NULL.

Note:
Complexity: O(1)

Precondition:
NODE1 != NULL & NODE2 != NULL

Parameters:
NODE1 The first low-level node to unlink from NODE2.

NODE2 The second low-level node to unlink from NODE1.

See also:
_gdsl_node_link()(p. 51)

3.4.2.12 void _gdsl_node_write (const _gdsl_node_t *NODE*, const gdsl_write_func_t *WRITE_F*, FILE * *OUTPUT_FILE*, void * *USER_DATA*)

Write the content of a low-level node to a file.

Write the content of the low-level node NODE to OUTPUT_FILE, using WRITE_F function. Additionnal USER_DATA argument could be passed to WRITE_F.

Note:
Complexity: O(1)

Precondition:
NODE != NULL & WRITE_F != NULL & OUTPUT_FILE != NULL

Parameters:

NODE The low-level node to write.

WRITE_F The write function.

OUTPUT_FILE The file where to write NODE's content.

USER_DATA User's datas passed to WRITE_F.

See also:

_gdsl_node_write_xml()(p. 54)
_gdsl_node_dump()(p. 49)

3.4.2.13 void _gdsl_node_write_xml (const _gdsl_node_t *NODE*, const gdsl_write_func_t *WRITE_F*, FILE * *OUTPUT_FILE*, void * *USER_DATA*)

Write the content of a low-level node to a file into XML.

Write the content of the low-level node NODE to OUTPUT_FILE, into XML language. If WRITE_F != NULL, then uses WRITE_F function to write NODE's content to OUTPUT_FILE. Additionnal USER_DATA argument could be passed to WRITE_F.

Note:

Complexity: O(1)

Precondition:

NODE != NULL & OUTPUT_FILE != NULL

Parameters:

NODE The low-level node to write.

WRITE_F The write function.

OUTPUT_FILE The file where to write NODE's content.

USER_DATA User's datas passed to WRITE_F.

See also:

_gdsl_node_write()(p. 53)
_gdsl_node_dump()(p. 49)

3.5 Main module

Functions

- const char * **gdsl_get_version** (void)
  Get GDSL version number as a string.

3.5.1 Function Documentation

3.5.1.1 const char* gdsl_get_version (void)

Get GDSL version number as a string.

Note:
Complexity: O(1)

Precondition:
nothing

Postcondition:
the returned string MUST NOT be deallocated.

Returns:
the GDSL version number as a string.

3.6 2D-Arrays manipulation module

Typedefs

- typedef gdsl_2darray * **gdsl_2darray_t**

  GDSL 2D-array type.

- typedef void(* **gdsl_2darray_write_func_t**)(**gdsl_element_t** E, const FILE *OUTPUT_FILE, **gdsl_2darray_position_t** POSITION, void *USER_DATA)

  GDSL 2D-array write function type.

Enumerations

- enum **gdsl_2darray_position_t** { **GDSL_2DARRAY_POSITION_FIRST_ROW** = 1, **GDSL_2DARRAY_POSITION_LAST_ROW** = 2, **GDSL_2DARRAY_POSITION_FIRST_COL** = 4, **GDSL_2DARRAY_POSITION_LAST_COL** = 8 }

  This type is for gdsl_2darray_write_func_t.

Functions

- **gdsl_2darray_t gdsl_2darray_alloc** (const char *NAME, const **ulong** R, const **ulong** C, const **gdsl_alloc_func_t** ALLOC_F, const **gdsl_free_func_t** FREE_F)

  Create a new 2D-array.

- void **gdsl_2darray_free** (**gdsl_2darray_t** A)

  Destroy a 2D-array.

- const char * **gdsl_2darray_get_name** (const **gdsl_2darray_t** A)

  Get the name of a 2D-array.

- **ulong gdsl_2darray_get_rows_number** (const **gdsl_2darray_t** A)

  Get the number of rows of a 2D-array.

- **ulong gdsl_2darray_get_columns_number** (const **gdsl_2darray_t** A)

  Get the number of columns of a 2D-array.

- **ulong gdsl_2darray_get_size** (const **gdsl_2darray_t** A)

  Get the size of a 2D-array.

- **gdsl_element_t gdsl_2darray_get_content** (const **gdsl_2darray_t** A, const **ulong** R, const **ulong** C)

  Get an element from a 2D-array.

- **gdsl_2darray_t gdsl_2darray_set_name** (**gdsl_2darray_t** A, const char *NEW_NAME)

  Set the name of a 2D-array.

- **gdsl_element_t gdsl_2darray_set_content** (**gdsl_2darray_t** A, const **ulong** R, const **ulong** C, void *VALUE)

  Modify an element in a 2D-array.

- void **gdsl_2darray_write** (const **gdsl_2darray_t** A, const **gdsl_2darray_write_func_t** WRITE_F, FILE *OUTPUT_FILE, void *USER_DATA)

  Write the content of a 2D-array to a file.

- void **gdsl_2darray_write_xml** (const **gdsl_2darray_t** A, const **gdsl_write_func_t** WRITE_F, FILE *OUTPUT_FILE, void *USER_DATA)

  Write the content of a 2D array to a file into XML.

- void **gdsl_2darray_dump** (const **gdsl_2darray_t** A, const **gdsl_write_func_t** WRITE_F, FILE *OUTPUT_FILE, void *USER_DATA)

  Dump the internal structure of a 2D array to a file.

3.6.1 Typedef Documentation

3.6.1.1 typedef struct gdsl_2darray* gdsl_2darray_t

GDSL 2D-array type.

This type is voluntary opaque. Variables of this kind could'nt be directly used, but by the functions of this module.

Definition at line 52 of file gdsl_2darray.h.

3.6.1.2 typedef void(* gdsl_2darray_write_func_t)(gdsl_element_t E, const FILE* OUTPUT_FILE, gdsl_2darray_position_t POSITION, void* USER_DATA)

GDSL 2D-array write function type.

Parameters:

E The gdsl_element_t variable to write

OUTPUT_FILE The file where to write E

POSITION is an or-ed combination of gdsl_2darray_position_t values to indicate where E is located into the gdsl_2darray_t mapped.

USER_DATA User's datas

Definition at line 82 of file gdsl_2darray.h.

3.6.2 Enumeration Type Documentation

3.6.2.1 enum gdsl_2darray_position_t

This type is for gdsl_2darray_write_func_t.

Enumeration values:

GDSL_2DARRAY_POSITION_FIRST_ROW When element is at first row

GDSL_2DARRAY_POSITION_LAST_ROW When element is at last row

GDSL_2DARRAY_POSITION_FIRST_COL When element is at first column

GDSL_2DARRAY_POSITION_LAST_COL When element is at last column

Definition at line 57 of file gdsl_2darray.h.

3.6.3 Function Documentation

3.6.3.1 gdsl_2darray_t gdsl_2darray_alloc (const char * *NAME*, const ulong *R*, const ulong *C*, const gdsl_alloc_func_t *ALLOC_F*, const gdsl_free_func_t *FREE_F*)

Create a new 2D-array.

Allocate a new 2D-array data structure with R rows and C columns and its name is set to a copy of NAME. The functions pointers ALLOC_F and FREE_F could be used to respectively, alloc and free elements in the 2D-array. These pointers could be set to NULL to use the default ones:

- the default ALLOC_F simply returns its argument
- the default FREE_F does nothing

Note:

Complexity: O(1)

Precondition:

nothing

Parameters:

NAME The name of the new 2D-array to create

R The number of rows of the new 2D-array to create

C The number of columns of the new 2D-array to create

ALLOC_F Function to alloc element when inserting it in a 2D-array

FREE_F Function to free element when removing it from a 2D-array

Returns:

the newly allocated 2D-array in case of success.
NULL in case of insufficient memory.

See also:

gdsl_2darray_free()(p. 60)
gdsl_alloc_func_t(p. 181)
gdsl_free_func_t(p. 182)

3.6.3.2 void gdsl_2darray_dump (const gdsl_2darray_t *A*, const gdsl_write_func_t *WRITE_F*, FILE * *OUTPUT_FILE*, void * *USER_DATA*)

Dump the internal structure of a 2D array to a file.

Dump A's structure to OUTPUT_FILE. If WRITE_F != NULL, then uses WRITE_F to write A's elements to OUTPUT_FILE. Additionnal USER_-DATA argument could be passed to WRITE_F.

Note:

Complexity: O(R x C), where R is A's rows count, and C is A's columns count

Precondition:

A must be a valid gdsl_2darray_t & OUTPUT_FILE != NULL

Parameters:

A The 2D-array to dump

WRITE_F The write function

OUTPUT_FILE The file where to write A's elements

USER_DATA User's datas passed to WRITE_F

See also:

gdsl_2darray_write()(p. 64)
gdsl_2darray_write_xml()(p. 64)

3.6.3.3 void gdsl_2darray_free (gdsl_2darray_t *A*)

Destroy a 2D-array.

Flush and destroy the 2D-array A. The FREE_F function passed to **gdsl_2darray_alloc()**(p. 58) is used to free elements from A, but no check is done to see if an element was set (ie. != NULL) or not.It's up to you to check if the element to free is NULL or not into the FREE_F function.

Note:
Complexity: O(R x C), where R is A's rows count, and C is A's columns count

Precondition:
A must be a valid gdsl_2darray_t

Parameters:
A The 2D-array to destroy

See also:
gdsl_2darray_alloc()(p. 58)

3.6.3.4 ulong gdsl_2darray_get_columns_number (const gdsl_2darray_t *A*)

Get the number of columns of a 2D-array.

Note:
Complexity: O(1)

Precondition:
A must be a valid gdsl_2darray_t

Parameters:
A The 2D-array from which getting the columns count

Returns:
the number of columns of the 2D-array A.

See also:
gdsl_2darray_get_rows_number()(p. 61)
gdsl_2darray_get_size()(p. 62)

3.6.3.5 gdsl_element_t gdsl_2darray_get_content (const gdsl_2darray_t *A*, const ulong *R*, const ulong *C*)

Get an element from a 2D-array.

Note:
Complexity: O(1)

Precondition:
A must be a valid gdsl_2darray_t & R <= gdsl_2darray_get_rows_number(A) & C <= gdsl_2darray_get_columns_number(A)

Parameters:
A The 2D-array from which getting the element

R The row indix of the element to get

C The column indix of the element to get

Returns:
the element of the 2D-array A contained in row R and column C.

See also:
gdsl_2darray_set_content()(p. 62)

3.6.3.6 const char* gdsl_2darray_get_name (const gdsl_2darray_t *A*)

Get the name of a 2D-array.

Note:
Complexity: O(1)

Precondition:
A must be a valid gdsl_2darray_t

Postcondition:
The returned string MUST NOT be freed.

Parameters:
A The 2D-array from which getting the name

Returns:
the name of the 2D-array A.

See also:
gdsl_2darray_set_name()(p. 63)

3.6.3.7 ulong gdsl_2darray_get_rows_number (const gdsl_2darray_t *A*)

Get the number of rows of a 2D-array.

Note:
Complexity: O(1)

Precondition:
A must be a valid gdsl_2darray_t

Parameters:
A The 2D-array from which getting the rows count

Returns:
the number of rows of the 2D-array A.

See also:
gdsl_2darray_get_columns_number()(p. 60)
gdsl_2darray_get_size()(p. 62)

3.6.3.8 ulong gdsl_2darray_get_size (const gdsl_2darray_t *A*)

Get the size of a 2D-array.

Note:
Complexity: O(1)

Precondition:
A must be a valid gdsl_2darray_t

Parameters:
A The 2D-array to use.

Returns:
the number of elements of A (noted |A|).

See also:
gdsl_2darray_get_rows_number()(p. 61)
gdsl_2darray_get_columns_number()(p. 60)

3.6.3.9 gdsl_element_t gdsl_2darray_set_content (gdsl_2darray_t *A*, const ulong *R*, const ulong *C*, void * *VALUE*)

Modify an element in a 2D-array.

Change the element at row R and column C of the 2D-array A, and returns it. The new element to insert is allocated using the ALLOC_F function passed to gdsl_2darray_create() applied on VALUE. The previous element contained in row R and in column C is NOT deallocated. It's up to you to do it before, if necessary.

Note:
Complexity: O(1)

Precondition:
A must be a valid gdsl_2darray_t & R <= gdsl_2darray_get_rows_number(A) & C <= gdsl_2darray_get_columns_number(A)

Parameters:
A The 2D-array to modify on element from

R The row number of the element to modify

C The column number of the element to modify

VALUE The user value to use for allocating the new element

Returns:
the newly allocated element in case of success.
NULL in case of insufficient memory.

See also:
gdsl_2darray_get_content()(p. 60)

3.6.3.10 gdsl_2darray_t gdsl_2darray_set_name (gdsl_2darray_t *A*, const char * *NEW_NAME*)

Set the name of a 2D-array.

Change the previous name of the 2D-array A to a copy of NEW_NAME.

Note:
Complexity: O(1)

Precondition:
A must be a valid gdsl_2darray_t

Parameters:
A The 2D-array to change the name

NEW_NAME The new name of A

Returns:
the modified 2D-array in case of success.
NULL in case of failure.

See also:
gdsl_2darray_get_name()(p. 61)

3.6.3.11 void gdsl_2darray_write (const gdsl_2darray_t *A*, const gdsl_2darray_write_func_t *WRITE_F*, FILE * *OUTPUT_FILE*, void * *USER_DATA*)

Write the content of a 2D-array to a file.

Write the elements of the 2D-array A to OUTPUT_FILE, using WRITE_F function. Additionnal USER_DATA argument could be passed to WRITE_F.

Note:
Complexity: O(R x C), where R is A's rows count, and C is A's columns count

Precondition:
WRITE_F != NULL & OUTPUT_FILE != NULL

Parameters:
A The 2D-array to write

WRITE_F The write function

OUTPUT_FILE The file where to write A's elements

USER_DATA User's datas passed to WRITE_F

See also:
gdsl_2darray_write_xml()(p. 64)
gdsl_2darray_dump()(p. 59)

3.6.3.12 void gdsl_2darray_write_xml (const gdsl_2darray_t *A*, const gdsl_write_func_t *WRITE_F*, FILE * *OUTPUT_FILE*, void * *USER_DATA*)

Write the content of a 2D array to a file into XML.

Write all A's elements to OUTPUT_FILE, into XML language. If WRITE_F != NULL, then uses WRITE_F to write A's elements to OUTPUT_FILE. Additionnal USER_DATA argument could be passed to WRITE_F.

Note:
Complexity: O(R x C), where R is A's rows count, and C is A's columns count

Precondition:
A must be a valid gdsl_2darray_t & OUTPUT_FILE != NULL

Parameters:
A The 2D-array to write

WRITE_F The write function

OUTPUT_FILE The file where to write A's elements

USER_DATA User's datas passed to WRITE_F

See also:
gdsl_2darray_write()(p. 64)
gdsl_2darray_dump()(p. 59)

3.7 Binary search tree manipulation module

Typedefs

- typedef gdsl_bstree * **gdsl_bstree_t**

  GDSL binary search tree type.

Functions

- **gdsl_bstree_t gdsl_bstree_alloc** (const char *NAME, **gdsl_alloc_func_t** ALLOC_F, **gdsl_free_func_t** FREE_F, **gdsl_compare_func_t** COMP_F)

  Create a new binary search tree.

- void **gdsl_bstree_free** (**gdsl_bstree_t** T)

  Destroy a binary search tree.

- void **gdsl_bstree_flush** (**gdsl_bstree_t** T)

  Flush a binary search tree.

- const char * **gdsl_bstree_get_name** (const **gdsl_bstree_t** T)

  Get the name of a binary search tree.

- **bool gdsl_bstree_is_empty** (const **gdsl_bstree_t** T)

  Check if a binary search tree is empty.

- **gdsl_element_t gdsl_bstree_get_root** (const **gdsl_bstree_t** T)

  Get the root of a binary search tree.

- **ulong gdsl_bstree_get_size** (const **gdsl_bstree_t** T)

  Get the size of a binary search tree.

- **ulong gdsl_bstree_get_height** (const **gdsl_bstree_t** T)

  Get the height of a binary search tree.

- **gdsl_bstree_t gdsl_bstree_set_name** (**gdsl_bstree_t** T, const char *NEW_NAME)

  Set the name of a binary search tree.

- **gdsl_element_t gdsl_bstree_insert** (**gdsl_bstree_t** T, void *VALUE, int *RESULT)

  Insert an element into a binary search tree if it's not found or return it.

- **gdsl_element_t gdsl_bstree_remove** (**gdsl_bstree_t** T, void *VALUE)

Remove an element from a binary search tree.

- **gdsl_bstree_t gdsl_bstree_delete** (**gdsl_bstree_t** T, void *VALUE)

  Delete an element from a binary search tree.

- **gdsl_element_t gdsl_bstree_search** (const **gdsl_bstree_t** T, **gdsl_compare_func_t** COMP_F, void *VALUE)

  Search for a particular element into a binary search tree.

- **gdsl_element_t gdsl_bstree_map_prefix** (const **gdsl_bstree_t** T, **gdsl_map_func_t** MAP_F, void *USER_DATA)

  Parse a binary search tree in prefixed order.

- **gdsl_element_t gdsl_bstree_map_infix** (const **gdsl_bstree_t** T, **gdsl_map_func_t** MAP_F, void *USER_DATA)

  Parse a binary search tree in infixed order.

- **gdsl_element_t gdsl_bstree_map_postfix** (const **gdsl_bstree_t** T, **gdsl_map_func_t** MAP_F, void *USER_DATA)

  Parse a binary search tree in postfixed order.

- void **gdsl_bstree_write** (const **gdsl_bstree_t** T, **gdsl_write_func_t** WRITE_F, FILE *OUTPUT_FILE, void *USER_DATA)

  Write the element of each node of a binary search tree to a file.

- void **gdsl_bstree_write_xml** (const **gdsl_bstree_t** T, **gdsl_write_func_t** WRITE_F, FILE *OUTPUT_FILE, void *USER_DATA)

  Write the content of a binary search tree to a file into XML.

- void **gdsl_bstree_dump** (const **gdsl_bstree_t** T, **gdsl_write_func_t** WRITE_F, FILE *OUTPUT_FILE, void *USER_DATA)

  Dump the internal structure of a binary search tree to a file.

3.7.1 Typedef Documentation

3.7.1.1 typedef struct gdsl_bstree* gdsl_bstree_t

GDSL binary search tree type.

This type is voluntary opaque. Variables of this kind could'nt be directly used, but by the functions of this module.

Definition at line 52 of file gdsl_bstree.h.

3.7.2 Function Documentation

3.7.2.1 gdsl_bstree_t gdsl_bstree_alloc (const char * *NAME*, gdsl_alloc_func_t *ALLOC_F*, gdsl_free_func_t *FREE_F*, gdsl_compare_func_t *COMP_F*)

Create a new binary search tree.

Allocate a new binary search tree data structure which name is set to a copy of NAME. The function pointers ALLOC_F, FREE_F and COMP_F could be used to respectively alloc, free and compares elements in the tree. These pointers could be set to NULL to use the default ones:

- the default ALLOC_F simply returns its argument
- the default FREE_F does nothing
- the default COMP_F always returns 0

Note:
Complexity: O(1)

Precondition:
nothing

Parameters:
NAME The name of the new binary tree to create
ALLOC_F Function to alloc element when inserting it in a binary tree
FREE_F Function to free element when removing it from a binary tree
COMP_F Function to compare elements into the binary tree

Returns:
the newly allocated binary search tree in case of success.
NULL in case of insufficient memory.

See also:
gdsl_bstree_free()(p. 70)
gdsl_bstree_flush()(p. 70)
gdsl_alloc_func_t(p. 181)
gdsl_free_func_t(p. 182)
gdsl_compare_func_t(p. 181)

3.7.2.2 gdsl_bstree_t gdsl_bstree_delete (gdsl_bstree_t *T*, void * *VALUE*)

Delete an element from a binary search tree.

Remove from the binary search tree the first founded element E equal to VALUE, by using T's COMP_F function passed to **gdsl_bstree_alloc()**(p. 68). If E is found, it is removed from T and E is deallocated using T's FREE_F function passed to **gdsl_bstree_alloc()**(p. 68), then T is returned.

Note:
Complexity: O(h(T)), where log2(|T|) <= h(T) <= |T|-1
the resulting T is modified by examinating the left sub-tree from the founded E.

Precondition:
T must be a valid gdsl_bstree_t

Parameters:
T The binary search tree to remove an element from

VALUE The value used to find the element to remove

Returns:
the modified binary search tree after removal of E if E was found.
NULL if no element equal to VALUE was found.

See also:
gdsl_bstree_insert()(p. 72)
gdsl_bstree_remove()(p. 75)

3.7.2.3 void gdsl_bstree_dump (const gdsl_bstree_t *T*, gdsl_write_func_t *WRITE_F*, FILE * *OUTPUT_FILE*, void * *USER_DATA*)

Dump the internal structure of a binary search tree to a file.

Dump the structure of the binary search tree T to OUTPUT_FILE. If WRITE_F != NULL, then use WRITE_F to write T's nodes elements to OUTPUT_FILE. Additionnal USER_DATA argument could be passed to WRITE_F.

Note:
Complexity: O(|T|)

Precondition:
T must be a valid gdsl_bstree_t & OUTPUT_FILE != NULL

Parameters:
T The binary search tree to write.

WRITE_F The write function.

OUTPUT_FILE The file where to write T's elements.

USER_DATA User's datas passed to WRITE_F.

See also:
gdsl_bstree_write()(p. 77)
gdsl_bstree_write_xml()(p. 77)

3.7.2.4 void gdsl_bstree_flush (gdsl_bstree_t *T*)

Flush a binary search tree.

Deallocate all the elements of the binary search tree T by calling T's FREE_F function passed to **gdsl_rbtree_alloc()**(p. 155). The binary search tree T is not deallocated itself and its name is not modified.

Note:
Complexity: O(|T|)

Precondition:
T must be a valid gdsl_bstree_t

Parameters:
T The binary search tree to flush

See also:
gdsl_bstree_alloc()(p. 68)
gdsl_bstree_free()(p. 70)

3.7.2.5 void gdsl_bstree_free (gdsl_bstree_t *T*)

Destroy a binary search tree.

Deallocate all the elements of the binary search tree T by calling T's FREE_F function passed to **gdsl_bstree_alloc()**(p. 68). The name of T is deallocated and T is deallocated itself too.

Note:
Complexity: O(|T|)

Precondition:
T must be a valid gdsl_bstree_t

Parameters:
T The binary search tree to deallocate

See also:
gdsl_bstree_alloc()(p. 68)
gdsl_bstree_flush()(p. 70)

3.7.2.6 ulong gdsl_bstree_get_height (const gdsl_bstree_t *T*)

Get the height of a binary search tree.

Note:
Complexity: O(|T|)

Precondition:
T must be a valid gdsl_bstree_t

Parameters:
T The binary search tree to compute the height from

Returns:
the height of the binary search tree T (noted h(T)).

See also:
gdsl_bstree_get_size()(p. 72)

3.7.2.7 const char* gdsl_bstree_get_name (const gdsl_bstree_t *T*)

Get the name of a binary search tree.

Note:
Complexity: O(1)

Precondition:
T must be a valid gdsl_bstree_t

Postcondition:
The returned string MUST NOT be freed.

Parameters:
T The binary search tree to get the name from

Returns:
the name of the binary search tree T.

See also:
gdsl_bstree_set_name(p. 76) ()

3.7.2.8 gdsl_element_t gdsl_bstree_get_root (const gdsl_bstree_t *T*)

Get the root of a binary search tree.

Note:
Complexity: O(1)

Precondition:
T must be a valid gdsl_bstree_t

Parameters:
T The binary search tree to get the root element from

Returns:
the element at the root of the binary search tree T.

3.7.2.9 ulong gdsl_bstree_get_size (const gdsl_bstree_t *T*)

Get the size of a binary search tree.

Note:
Complexity: O(1)

Precondition:
T must be a valid gdsl_bstree_t

Parameters:
T The binary search tree to get the size from

Returns:
the size of the binary search tree T (noted |T|).

See also:
gdsl_bstree_get_height()(p. 70)

3.7.2.10 gdsl_element_t gdsl_bstree_insert (gdsl_bstree_t *T*, void * *VALUE*, int * *RESULT*)

Insert an element into a binary search tree if it's not found or return it.

Search for the first element E equal to VALUE into the binary search tree T, by using T's COMP_F function passed to gdsl_bstree_alloc to find it. If E is found, then it's returned. If E isn't found, then a new element E is allocated using T's ALLOC_F function passed to gdsl_bstree_alloc and is inserted and then returned.

Note:
Complexity: O(h(T)), where log2(|T|) <= h(T) <= |T|-1

Precondition:
T must be a valid gdsl_bstree_t & RESULT != NULL

Parameters:
T The binary search tree to modify
VALUE The value used to make the new element to insert into T
RESULT The address where the result code will be stored.

Returns:
the element E and RESULT = GDSL_OK if E is inserted into T.
the element E and RESULT = GDSL_ERR_DUPLICATE_ENTRY if E is already present in T.
NULL and RESULT = GDSL_ERR_MEM_ALLOC in case of insufficient memory.

See also:
gdsl_bstree_remove()(p. 75)
gdsl_bstree_delete()(p. 68)

3.7.2.11 bool gdsl_bstree_is_empty (const gdsl_bstree_t *T*)

Check if a binary search tree is empty.

Note:
Complexity: O(1)

Precondition:
T must be a valid gdsl_bstree_t

Parameters:
T The binary search tree to check

Returns:
TRUE if the binary search tree T is empty.
FALSE if the binary search tree T is not empty.

3.7.2.12 gdsl_element_t gdsl_bstree_map_infix (const gdsl_bstree_t *T*, gdsl_map_func_t *MAP_F*, void * *USER_DATA*)

Parse a binary search tree in infixed order.

Parse all nodes of the binary search tree T in infixed order. The MAP_F function is called on the element contained in each node with the USER_DATA argument. If MAP_F returns GDSL_MAP_STOP, then **gdsl_bstree_map_infix()**(p. 73) stops and returns its last examinated element.

Note:
Complexity: O(|T|)

Precondition:
T must be a valid gdsl_bstree_t & MAP_F != NULL

Parameters:
T The binary search tree to map.
MAP_F The map function.
USER_DATA User's datas passed to MAP_F

Returns:
the first element for which MAP_F returns GDSL_MAP_STOP.
NULL when the parsing is done.

See also:
gdsl_bstree_map_prefix()(p. 74)
gdsl_bstree_map_postfix()(p. 74)

3.7.2.13 gdsl_element_t gdsl_bstree_map_postfix (const gdsl_bstree_t *T*, gdsl_map_func_t *MAP_F*, void * *USER_DATA*)

Parse a binary search tree in postfixed order.

Parse all nodes of the binary search tree T in postfixed order. The MAP_F function is called on the element contained in each node with the USER_DATA argument. If MAP_F returns GDSL_MAP_STOP, then **gdsl_bstree_map_postfix()**(p. 74) stops and returns its last examinated element.

Note:
Complexity: O(|T|)

Precondition:
T must be a valid gdsl_bstree_t & MAP_F != NULL

Parameters:
T The binary search tree to map.

MAP_F The map function.

USER_DATA User's datas passed to MAP_F

Returns:
the first element for which MAP_F returns GDSL_MAP_STOP.
NULL when the parsing is done.

See also:
gdsl_bstree_map_prefix()(p. 74)
gdsl_bstree_map_infix()(p. 73)

3.7.2.14 gdsl_element_t gdsl_bstree_map_prefix (const gdsl_bstree_t *T*, gdsl_map_func_t *MAP_F*, void * *USER_DATA*)

Parse a binary search tree in prefixed order.

Parse all nodes of the binary search tree T in prefixed order. The MAP_F function is called on the element contained in each node with the USER_DATA argument. If MAP_F returns GDSL_MAP_STOP, then **gdsl_bstree_map_prefix()**(p. 74) stops and returns its last examinated element.

Note:
Complexity: O(|T|)

Precondition:
T must be a valid gdsl_bstree_t & MAP_F != NULL

Parameters:
T The binary search tree to map.

MAP_F The map function.

USER_DATA User's datas passed to MAP_F

Returns:
the first element for which MAP_F returns GDSL_MAP_STOP.
NULL when the parsing is done.

See also:
gdsl_bstree_map_infix()(p. 73)
gdsl_bstree_map_postfix()(p. 74)

3.7.2.15 gdsl_element_t gdsl_bstree_remove (gdsl_bstree_t *T*, void * *VALUE*)

Remove an element from a binary search tree.

Remove from the binary search tree T the first founded element E equal to VALUE, by using T's COMP_F function passed to **gdsl_bstree_alloc()**(p. 68). If E is found, it is removed from T and then returned.

Note:
Complexity: O(h(T)), where log2(|T|) <= h(T) <= |T|-1
The resulting T is modified by examinating the left sub-tree from the founded E.

Precondition:
T must be a valid gdsl_bstree_t

Parameters:
T The binary search tree to modify

VALUE The value used to find the element to remove

Returns:
the first founded element equal to VALUE in T in case is found.
NULL in case no element equal to VALUE is found in T.

See also:
gdsl_bstree_insert()(p. 72)
gdsl_bstree_delete()(p. 68)

3.7.2.16 gdsl_element_t gdsl_bstree_search (const gdsl_bstree_t *T*, gdsl_compare_func_t *COMP_F*, void * *VALUE*)

Search for a particular element into a binary search tree.

Search the first element E equal to VALUE in the binary seach tree T, by using COMP_F function to find it. If COMP_F == NULL, then the COMP_F function passed to **gdsl_bstree_alloc()**(p. 68) is used.

Note:
Complexity: O(h(T)), where log2(|T|) <= h(T) <= |T|-1

Precondition:
T must be a valid gdsl_bstree_t

Parameters:
T The binary search tree to use.

COMP_F The comparison function to use to compare T's element with VALUE to find the element E (or NULL to use the default T's COMP_F)

VALUE The value that must be used by COMP_F to find the element E

Returns:
the first founded element E equal to VALUE.
NULL if VALUE is not found in T.

See also:
gdsl_bstree_insert()(p. 72)
gdsl_bstree_remove()(p. 75)
gdsl_bstree_delete()(p. 68)

3.7.2.17 gdsl_bstree_t gdsl_bstree_set_name (gdsl_bstree_t *T*, const char * *NEW_NAME*)

Set the name of a binary search tree.

Change the previous name of the binary search tree T to a copy of NEW_NAME.

Note:
Complexity: O(1)

Precondition:
T must be a valid gdsl_bstree_t

Parameters:
T The binary search tree to change the name

NEW_NAME The new name of T

Returns:
the modified binary search tree in case of success.
NULL in case of insufficient memory.

See also:
gdsl_bstree_get_name()(p. 71)

3.7.2.18 void gdsl_bstree_write (const gdsl_bstree_t *T*, gdsl_write_func_t *WRITE_F*, FILE * *OUTPUT_FILE*, void * *USER_DATA*)

Write the element of each node of a binary search tree to a file.

Write the nodes elements of the binary search tree T to OUTPUT_FILE, using WRITE_F function. Additionnal USER_DATA argument could be passed to WRITE_F.

Note:
Complexity: O(|T|)

Precondition:
T must be a valid gdsl_bstree_t & WRITE_F != NULL & OUTPUT_FILE != NULL

Parameters:
T The binary search tree to write.

WRITE_F The write function.

OUTPUT_FILE The file where to write T's elements.

USER_DATA User's datas passed to WRITE_F.

See also:
gdsl_bstree_write_xml()(p. 77)
gdsl_bstree_dump()(p. 69)

3.7.2.19 void gdsl_bstree_write_xml (const gdsl_bstree_t *T*, gdsl_write_func_t *WRITE_F*, FILE * *OUTPUT_FILE*, void * *USER_DATA*)

Write the content of a binary search tree to a file into XML.

Write the nodes elements of the binary search tree T to OUTPUT_FILE, into XML language. If WRITE_F != NULL, then use WRITE_F to write T's nodes elements to OUTPUT_FILE. Additionnal USER_DATA argument could be passed to WRITE_F.

Note:
Complexity: O(|T|)

Precondition:
T must be a valid gdsl_bstree_t & OUTPUT_FILE != NULL

Parameters:
T The binary search tree to write.

WRITE_F The write function.

OUTPUT_FILE The file where to write T's elements.

USER_DATA User's datas passed to WRITE_F.

See also:

gdsl_bstree_write()(p. 77)
gdsl_bstree_dump()(p. 69)

3.8 Hashtable manipulation module

Typedefs

- typedef hash_table * **gdsl_hash_t**
  GDSL hashtable type.
- typedef const char *(* **gdsl_key_func_t**)(void *VALUE)
  GDSL hashtable key function type.
- typedef const **ulong**(* **gdsl_hash_func_t**)(const char *KEY)
  GDSL hashtable hash function type.

Functions

- const **ulong gdsl_hash** (const char *KEY)
  Computes a hash value from a NULL terminated character string.
- **gdsl_hash_t gdsl_hash_alloc** (const char *NAME, **gdsl_alloc_func_t** ALLOC_F, **gdsl_free_func_t** FREE_F, **gdsl_key_func_t** KEY_F, **gdsl_hash_func_t** HASH_F, ushort INITIAL_ENTRIES_NB)
  Create a new hashtable.
- void **gdsl_hash_free** (**gdsl_hash_t** H)
  Destroy a hashtable.
- void **gdsl_hash_flush** (**gdsl_hash_t** H)
  Flush a hashtable.
- const char * **gdsl_hash_get_name** (const **gdsl_hash_t** H)
  Get the name of a hashtable.
- ushort **gdsl_hash_get_entries_number** (const **gdsl_hash_t** H)
  Get the number of entries of a hashtable.
- ushort **gdsl_hash_get_lists_max_size** (const **gdsl_hash_t** H)
  Get the max number of elements allowed in each entry of a hashtable.
- ushort **gdsl_hash_get_longest_list_size** (const **gdsl_hash_t** H)
  Get the number of elements of the longest list entry of a hashtable.
- **ulong gdsl_hash_get_size** (const **gdsl_hash_t** H)
  Get the size of a hashtable.

- double **gdsl_hash_get_fill_factor** (const **gdsl_hash_t** H)

  Get the fill factor of a hashtable.

- **gdsl_hash_t gdsl_hash_set_name** (**gdsl_hash_t** H, const char *NEW_NAME)

  Set the name of a hashtable.

- **gdsl_element_t gdsl_hash_insert** (**gdsl_hash_t** H, void *VALUE)

  Insert an element into a hashtable (PUSH).

- **gdsl_element_t gdsl_hash_remove** (**gdsl_hash_t** H, const char *KEY)

  Remove an element from a hashtable (POP).

- **gdsl_hash_t gdsl_hash_delete** (**gdsl_hash_t** H, const char *KEY)

  Delete an element from a hashtable.

- **gdsl_hash_t gdsl_hash_modify** (**gdsl_hash_t** H, ushort NEW_ENTRIES_NB, ushort NEW_LISTS_MAX_SIZE)

  Increase the dimensions of a hashtable.

- **gdsl_element_t gdsl_hash_search** (const **gdsl_hash_t** H, const char *KEY)

  Search for a particular element into a hashtable (GET).

- **gdsl_element_t gdsl_hash_map** (const **gdsl_hash_t** H, **gdsl_map_func_t** MAP_F, void *USER_DATA)

  Parse a hashtable.

- void **gdsl_hash_write** (const **gdsl_hash_t** H, **gdsl_write_func_t** WRITE_F, FILE *OUTPUT_FILE, void *USER_DATA)

  Write all the elements of a hashtable to a file.

- void **gdsl_hash_write_xml** (const **gdsl_hash_t** H, **gdsl_write_func_t** WRITE_F, FILE *OUTPUT_FILE, void *USER_DATA)

  Write the content of a hashtable to a file into XML.

- void **gdsl_hash_dump** (const **gdsl_hash_t** H, **gdsl_write_func_t** WRITE_F, FILE *OUTPUT_FILE, void *USER_DATA)

  Dump the internal structure of a hashtable to a file.

3.8.1 Typedef Documentation

3.8.1.1 typedef const ulong(* gdsl_hash_func_t)(const char* KEY)

GDSL hashtable hash function type.

Parameters:
KEY the key used to compute the hash code.

Returns:
The hashed value computed from KEY.

Definition at line 69 of file gdsl_hash.h.

3.8.1.2 typedef struct hash_table* gdsl_hash_t

GDSL hashtable type.

This type is voluntary opaque. Variables of this kind could'nt be directly used, but by the functions of this module.

Definition at line 53 of file gdsl_hash.h.

3.8.1.3 typedef const char*(* gdsl_key_func_t)(void* VALUE)

GDSL hashtable key function type.

Postcondition:
Returned value must be != "" && != NULL.

Parameters:
VALUE The value used to get the key from

Returns:
The key associated to the VALUE.

Definition at line 61 of file gdsl_hash.h.

3.8.2 Function Documentation

3.8.2.1 const ulong gdsl_hash (const char * *KEY*)

Computes a hash value from a NULL terminated character string.

This function computes a hash value from the NULL terminated KEY string.

Note:
Complexity: O (|key|)

Precondition:
KEY must be NULL-terminated.

Parameters:
KEY The NULL terminated string to compute the key from

Returns:
the hash code computed from KEY.

3.8.2.2 gdsl_hash_t gdsl_hash_alloc (const char * *NAME*, gdsl_alloc_func_t *ALLOC_F*, gdsl_free_func_t *FREE_F*, gdsl_key_func_t *KEY_F*, gdsl_hash_func_t *HASH_F*, ushort *INITIAL_ENTRIES_NB*)

Create a new hashtable.

Allocate a new hashtable data structure which name is set to a copy of NAME. The new hashtable will contain initially INITIAL_ENTRIES_NB lists. This value could be (only) increased with **gdsl_hash_modify()**(p. 89) function. Until this function is called, then all H's lists entries have no size limit. The function pointers ALLOC_F and FREE_F could be used to respectively, alloc and free elements in the hashtable. The KEY_F function must provide a unique key associated to its argument. The HASH_F function must compute a hash code from its argument. These pointers could be set to NULL to use the default ones:

- the default ALLOC_F simply returns its argument
- the default FREE_F does nothing
- the default KEY_F simply returns its argument
- the default HASH_F is **gdsl_hash()**(p. 81) above

Note:
Complexity: O(1)

Precondition:
nothing.

Parameters:
NAME The name of the new hashtable to create

ALLOC_F Function to alloc element when inserting it in the hashtable

FREE_F Function to free element when deleting it from the hashtable

KEY_F Function to get the key from an element

HASH_F Function used to compute the hash value.

INITIAL_ENTRIES_NB Initial number of entries of the hashtable

Returns:
the newly allocated hashtable in case of success.
NULL in case of insufficient memory.

See also:
gdsl_hash_free()(p. 84)
gdsl_hash_flush()(p. 84)
gdsl_hash_insert()(p. 88)
gdsl_hash_modify()(p. 89)

3.8.2.3 gdsl_hash_t gdsl_hash_delete (gdsl_hash_t *H*, const char * *KEY*)

Delete an element from a hashtable.

Remove from he hashtable H the first founded element E equal to KEY. If E is found, it is removed from H and E is deallocated using H's FREE_F function passed to **gdsl_hash_alloc()**(p. 82), then H is returned.

Note:
Complexity: O(M), where M is the average size of H's lists

Precondition:
H must be a valid gdsl_hash_t

Parameters:
H The hashtable to modify

KEY The key used to find the element to remove

Returns:
the modified hashtable after removal of E if E was found.
NULL if no element equal to KEY was found.

See also:
gdsl_hash_insert()(p. 88)
gdsl_hash_search()(p. 90)
gdsl_hash_remove()(p. 90)

3.8.2.4 void gdsl_hash_dump (const gdsl_hash_t *H*, gdsl_write_func_t *WRITE_F*, FILE * *OUTPUT_FILE*, void * *USER_DATA*)

Dump the internal structure of a hashtable to a file.

Dump the structure of the hashtable H to OUTPUT_FILE. If WRITE_F != NULL, then uses WRITE_F to write H's elements to OUTPUT_FILE. Additionnal USER_DATA argument could be passed to WRITE_F.

Note:
Complexity: O(|H|)

Precondition:
H must be a valid gdsl_hash_t & OUTPUT_FILE != NULL

Parameters:
H The hashtable to write

WRITE_F The write function

OUTPUT_FILE The file where to write H's elements

USER_DATA User's datas passed to WRITE_F

See also:
gdsl_hash_write()(p. 92)
gdsl_hash_write_xml()(p. 92)

3.8.2.5 void gdsl_hash_flush (gdsl_hash_t *H*)

Flush a hashtable.

Deallocate all the elements of the hashtable H by calling H's FREE_F function passed to **gdsl_hash_alloc()**(p. 82). H is not deallocated itself and H's name is not modified.

Note:
Complexity: O(|H|)

Precondition:
H must be a valid gdsl_hash_t

Parameters:
H The hashtable to flush

See also:
gdsl_hash_alloc()(p. 82)
gdsl_hash_free()(p. 84)

3.8.2.6 void gdsl_hash_free (gdsl_hash_t *H*)

Destroy a hashtable.

Deallocate all the elements of the hashtable H by calling H's FREE_F function passed to **gdsl_hash_alloc()**(p. 82). The name of H is deallocated and H is deallocated itself too.

Note:
Complexity: O(|H|)

Precondition:
H must be a valid gdsl_hash_t

Parameters:
H The hashtable to destroy

See also:
gdsl_hash_alloc()(p. 82)
gdsl_hash_flush()(p. 84)

3.8.2.7 ushort gdsl_hash_get_entries_number (const gdsl_hash_t *H*)

Get the number of entries of a hashtable.

Note:
Complexity: O(1)

Precondition:
H must be a valid gdsl_hash_t

Parameters:
H The hashtable to use.

Returns:
the number of lists entries of the hashtable H.

See also:
gdsl_hash_get_size()(p. 87)
gdsl_hash_fill_factor()

3.8.2.8 double gdsl_hash_get_fill_factor (const gdsl_hash_t *H*)

Get the fill factor of a hashtable.

Note:
Complexity: O(L), where L = gdsl_hash_get_entries_number(H)

Precondition:
H must be a valid gdsl_hash_t

Parameters:
H The hashtable to use

Returns:
The fill factor of H, computed as |H| / L

See also:
gdsl_hash_get_entries_number()(p. 85)
gdsl_hash_get_longest_list_size()(p. 86)
gdsl_hash_get_size()(p. 87)

3.8.2.9 ushort gdsl_hash_get_lists_max_size (const gdsl_hash_t *H*)

Get the max number of elements allowed in each entry of a hashtable.

Note:
Complexity: O(1)

Precondition:
H must be a valid gdsl_hash_t

Parameters:
H The hashtable to use.

Returns:
0 if no lists max size was set before (ie. no limit for H's entries).
the max number of elements for each entry of the hashtable H, if the function **gdsl_hash_modify()**(p. 89) was used with a NEW_LISTS_MAX_SIZE greather than the actual one.

See also:
gdsl_hash_fill_factor()
gdsl_hash_get_entries_number()(p. 85)
gdsl_hash_get_longest_list_size()(p. 86)
gdsl_hash_modify()(p. 89)

3.8.2.10 ushort gdsl_hash_get_longest_list_size (const gdsl_hash_t *H*)

Get the number of elements of the longest list entry of a hashtable.

Note:
Complexity: O(L), where L = gdsl_hash_get_entries_number(H)

Precondition:
H must be a valid gdsl_hash_t

Parameters:
H The hashtable to use.

Returns:
the number of elements of the longest list entry of the hashtable H.

See also:
gdsl_hash_get_size()(p. 87)
gdsl_hash_fill_factor()
gdsl_hash_get_entries_number()(p. 85)
gdsl_hash_get_lists_max_size()(p. 86)

3.8.2.11 const char* gdsl_hash_get_name (const gdsl_hash_t *H*)

Get the name of a hashtable.

Note:
Complexity: O(1)

Precondition:
H must be a valid gdsl_hash_t

Postcondition:
The returned string MUST NOT be freed.

Parameters:
H The hashtable to get the name from

Returns:
the name of the hashtable H.

See also:
gdsl_hash_set_name()(p. 91)

3.8.2.12 ulong gdsl_hash_get_size (const gdsl_hash_t *H*)

Get the size of a hashtable.

Note:
Complexity: O(L), where L = gdsl_hash_get_entries_number(H)

Precondition:
H must be a valid gdsl_hash_t

Parameters:
H The hashtable to get the size from

Returns:
the number of elements of H (noted |H|).

See also:
gdsl_hash_get_entries_number()(p. 85)
gdsl_hash_fill_factor()
gdsl_hash_get_longest_list_size()(p. 86)

3.8.2.13 gdsl_element_t gdsl_hash_insert (gdsl_hash_t *H*, void * *VALUE*)

Insert an element into a hashtable (PUSH).

Allocate a new element E by calling H's ALLOC_F function on VALUE. The key K of the new element E is computed using KEY_F called on E. If the value of gdsl_hash_get_lists_max_size(H) is not reached, or if it is equal to zero, then the insertion is simple. Otherwise, H is re-organized as follow:

- its actual gdsl_hash_get_entries_number(H) (say N) is modified as N * 2 + 1
- its actual gdsl_hash_get_lists_max_size(H) (say M) is modified as M * 2 The element E is then inserted into H at the entry computed by HASH_F(K) modulo gdsl_hash_get_entries_number(H). ALLOC_F, KEY_F and HASH_F are the function pointers passed to **gdsl_hash_alloc()**(p. 82).

Note:
Complexity: O(1) if gdsl_hash_get_lists_max_size(H) is not reached or if it is equal to zero
Complexity: O (gdsl_hash_modify (H)) if gdsl_hash_get_lists_max_size(H) is reached, so H needs to grow

Precondition:
H must be a valid gdsl_hash_t

Parameters:
H The hashtable to modify

VALUE The value used to make the new element to insert into H

Returns:
the inserted element E in case of success.
NULL in case of insufficient memory.

See also:
gdsl_hash_alloc()(p. 82)
gdsl_hash_remove()(p. 90)
gdsl_hash_delete()(p. 83)
gdsl_hash_get_size()(p. 87)
gdsl_hash_get_entries_number()(p. 85)
gdsl_hash_modify()(p. 89)

3.8.2.14 gdsl_element_t gdsl_hash_map (const gdsl_hash_t *H*, gdsl_map_func_t *MAP_F*, void * *USER_DATA*)

Parse a hashtable.

Parse all elements of the hashtable H. The MAP_F function is called on each H's element with USER_DATA argument. If MAP_F returns GDSL_MAP_-STOP then **gdsl_hash_map()**(p. 88) stops and returns its last examinated element.

Note:
Complexity: O(|H|)

Precondition:
H must be a valid gdsl_hash_t & MAP_F != NULL

Parameters:
H The hashtable to map

MAP_F The map function.

USER_DATA User's datas passed to MAP_F

Returns:
the first element for which MAP_F returns GDSL_MAP_STOP.
NULL when the parsing is done.

3.8.2.15 gdsl_hash_t gdsl_hash_modify (gdsl_hash_t *H*, ushort *NEW_ENTRIES_NB*, ushort *NEW_LISTS_MAX_SIZE*)

Increase the dimensions of a hashtable.

The hashtable H is re-organized to have NEW_ENTRIES_NB lists entries. Each entry is limited to NEW_LISTS_MAX_SIZE elements. After a call to this function, all insertions into H will make H automatically growing if needed. The grow is needed each time an insertion makes an entry list to reach NEW_-LISTS_MAX_SIZE elements. In this case, H will be reorganized automatically by **gdsl_hash_insert()**(p. 88).

Note:
Complexity: O(|H|)

Precondition:
H must be a valid gdsl_hash_t & NEW_ENTRIES_NB > gdsl_hash_-get_entries_number(H) & NEW_LISTS_MAX_SIZE > gdsl_hash_-get_lists_max_size(H)

Parameters:
H The hashtable to modify

NEW_ENTRIES_NB

NEW_LISTS_MAX_SIZE

Returns:
the modified hashtable H in case of success
NULL in case of failure, or in case NEW_ENTRIES_NB <= gdsl_hash_get_entries_number(H) or in case NEW_LISTS_MAX_SIZE <= gdsl_hash_get_lists_max_size(H) in these cases, H is not modified

See also:
gdsl_hash_insert()(p. 88)
gdsl_hash_get_entries_number()(p. 85)
gdsl_hash_get_fill_factor()(p. 85)
gdsl_hash_get_longest_list_size()(p. 86)
gdsl_hash_get_lists_max_size()(p. 86)

3.8.2.16 gdsl_element_t gdsl_hash_remove (gdsl_hash_t *H*, const char * *KEY*)

Remove an element from a hashtable (POP).

Search into the hashtable H for the first element E equal to KEY. If E is found, it is removed from H and then returned.

Note:
Complexity: O(M), where M is the average size of H's lists

Precondition:
H must be a valid gdsl_hash_t

Parameters:
H The hashtable to modify

KEY The key used to find the element to remove

Returns:
the first founded element equal to KEY in H in case is found.
NULL in case no element equal to KEY is found in H.

See also:
gdsl_hash_insert()(p. 88)
gdsl_hash_search()(p. 90)
gdsl_hash_delete()(p. 83)

3.8.2.17 gdsl_element_t gdsl_hash_search (const gdsl_hash_t *H*, const char * *KEY*)

Search for a particular element into a hashtable (GET).

Search the first element E equal to KEY in the hashtable H.

Note:
Complexity: O(M), where M is the average size of H's lists

Precondition:
H must be a valid gdsl_hash_t

Parameters:
H The hashtable to search the element in

KEY The key to compare H's elements with

Returns:
the founded element E if it was found.
NULL in case the searched element E was not found.

See also:
gdsl_hash_insert()(p. 88)
gdsl_hash_remove()(p. 90)
gdsl_hash_delete()(p. 83)

3.8.2.18 gdsl_hash_t gdsl_hash_set_name (gdsl_hash_t *H*, const char * *NEW_NAME*)

Set the name of a hashtable.

Change the previous name of the hashtable H to a copy of NEW_NAME.

Note:
Complexity: O(1)

Precondition:
H must be a valid gdsl_hash_t

Parameters:
H The hashtable to change the name

NEW_NAME The new name of H

Returns:
the modified hashtable in case of success.
NULL in case of insufficient memory.

See also:
gdsl_hash_get_name()(p. 87)

3.8.2.19 void gdsl_hash_write (const gdsl_hash_t *H*, gdsl_write_func_t *WRITE_F*, FILE * *OUTPUT_FILE*, void * *USER_DATA*)

Write all the elements of a hashtable to a file.

Write the elements of the hashtable H to OUTPUT_FILE, using WRITE_F function. Additionnal USER_DATA argument could be passed to WRITE_F.

Note:
Complexity: O(|H|)

Precondition:
H must be a valid gdsl_hash_t & OUTPUT_FILE != NULL & WRITE_F != NULL

Parameters:
H The hashtable to write.

WRITE_F The write function.

OUTPUT_FILE The file where to write H's elements.

USER_DATA User's datas passed to WRITE_F.

See also:
gdsl_hash_write_xml()(p. 92)
gdsl_hash_dump()(p. 83)

3.8.2.20 void gdsl_hash_write_xml (const gdsl_hash_t *H*, gdsl_write_func_t *WRITE_F*, FILE * *OUTPUT_FILE*, void * *USER_DATA*)

Write the content of a hashtable to a file into XML.

Write the elements of the hashtable H to OUTPUT_FILE, into XML language. If WRITE_F != NULL, then uses WRITE_F to write H's elements to OUTPUT_FILE. Additionnal USER_DATA argument could be passed to WRITE_F.

Note:
Complexity: O(|H|)

Precondition:
H must be a valid gdsl_hash_t & OUTPUT_FILE != NULL

Parameters:
H The hashtable to write.

WRITE_F The write function.

OUTPUT_FILE The file where to write H's elements.

USER_DATA User's datas passed to WRITE_F.

See also:
gdsl_hash_write()(p. 92)
gdsl_hash_dump()(p. 83)

3.9 Doubly-linked list manipulation module

Typedefs

- typedef _gdsl_list * **gdsl_list_t**
  GDSL doubly-linked list type.

- typedef _gdsl_list_cursor * **gdsl_list_cursor_t**
  GDSL doubly-linked list cursor type.

Functions

- **gdsl_list_t gdsl_list_alloc** (const char *NAME, **gdsl_alloc_func_t** ALLOC_F, **gdsl_free_func_t** FREE_F)
  Create a new list.

- void **gdsl_list_free** (**gdsl_list_t** L)
  Destroy a list.

- void **gdsl_list_flush** (**gdsl_list_t** L)
  Flush a list.

- const char * **gdsl_list_get_name** (const **gdsl_list_t** L)
  Get the name of a list.

- **ulong gdsl_list_get_size** (const **gdsl_list_t** L)
  Get the size of a list.

- **bool gdsl_list_is_empty** (const **gdsl_list_t** L)
  Check if a list is empty.

- **gdsl_element_t gdsl_list_get_head** (const **gdsl_list_t** L)
  Get the head of a list.

- **gdsl_element_t gdsl_list_get_tail** (const **gdsl_list_t** L)
  Get the tail of a list.

- **gdsl_list_t gdsl_list_set_name** (**gdsl_list_t** L, const char *NEW_NAME)
  Set the name of a list.

- **gdsl_element_t gdsl_list_insert_head** (**gdsl_list_t** L, void *VALUE)
  Insert an element at the head of a list.

- **gdsl_element_t gdsl_list_insert_tail** (**gdsl_list_t** L, void *VALUE)

  Insert an element at the tail of a list.

- **gdsl_element_t gdsl_list_remove_head** (**gdsl_list_t** L)

  Remove the head of a list.

- **gdsl_element_t gdsl_list_remove_tail** (**gdsl_list_t** L)

  Remove the tail of a list.

- **gdsl_element_t gdsl_list_remove** (**gdsl_list_t** L, **gdsl_compare_func_t** COMP_F, const void *VALUE)

  Remove a particular element from a list.

- **gdsl_list_t gdsl_list_delete_head** (**gdsl_list_t** L)

  Delete the head of a list.

- **gdsl_list_t gdsl_list_delete_tail** (**gdsl_list_t** L)

  Delete the tail of a list.

- **gdsl_list_t gdsl_list_delete** (**gdsl_list_t** L, **gdsl_compare_func_t** COMP_F, const void *VALUE)

  Delete a particular element from a list.

- **gdsl_element_t gdsl_list_search** (const **gdsl_list_t** L, **gdsl_compare_func_t** COMP_F, const void *VALUE)

  Search for a particular element into a list.

- **gdsl_element_t gdsl_list_search_by_position** (const **gdsl_list_t** L, **ulong** POS)

  Search for an element by its position in a list.

- **gdsl_element_t gdsl_list_search_max** (const **gdsl_list_t** L, **gdsl_compare_func_t** COMP_F)

  Search for the greatest element of a list.

- **gdsl_element_t gdsl_list_search_min** (const **gdsl_list_t** L, **gdsl_compare_func_t** COMP_F)

  Search for the lowest element of a list.

- **gdsl_list_t gdsl_list_sort** (**gdsl_list_t** L, **gdsl_compare_func_t** COMP_F, **gdsl_element_t** MAX)

  Sort a list.

- **gdsl_element_t gdsl_list_map_forward** (const **gdsl_list_t** L, **gdsl_map_func_t** MAP_F, void *USER_DATA)

  Parse a list from head to tail.

- **gdsl_element_t gdsl_list_map_backward** (const **gdsl_list_t** L, **gdsl_map_func_t** MAP_F, void *USER_DATA)

  Parse a list from tail to head.

- void **gdsl_list_write** (const **gdsl_list_t** L, **gdsl_write_func_t** WRITE_F, FILE *OUTPUT_FILE, void *USER_DATA)

  Write all the elements of a list to a file.

- void **gdsl_list_write_xml** (const **gdsl_list_t** L, **gdsl_write_func_t** WRITE_F, FILE *OUTPUT_FILE, void *USER_DATA)

  Write the content of a list to a file into XML.

- void **gdsl_list_dump** (const **gdsl_list_t** L, **gdsl_write_func_t** WRITE_F, FILE *OUTPUT_FILE, void *USER_DATA)

  Dump the internal structure of a list to a file.

- **gdsl_list_cursor_t gdsl_list_cursor_alloc** (const **gdsl_list_t** L)

  Create a new list cursor.

- void **gdsl_list_cursor_free** (**gdsl_list_cursor_t** C)

  Destroy a list cursor.

- void **gdsl_list_cursor_move_to_head** (**gdsl_list_cursor_t** C)

  Put a cursor on the head of its list.

- void **gdsl_list_cursor_move_to_tail** (**gdsl_list_cursor_t** C)

  Put a cursor on the tail of its list.

- **gdsl_element_t gdsl_list_cursor_move_to_value** (**gdsl_list_cursor_t** C, **gdsl_compare_func_t** COMP_F, void *VALUE)

  Place a cursor on a particular element.

- **gdsl_element_t gdsl_list_cursor_move_to_position** (**gdsl_list_cursor_t** C, **ulong** POS)

  Place a cursor on a element given by its position.

- void **gdsl_list_cursor_step_forward** (**gdsl_list_cursor_t** C)

  Move a cursor one step forward of its list.

- void **gdsl_list_cursor_step_backward** (**gdsl_list_cursor_t** C)

  Move a cursor one step backward of its list.

- **bool gdsl_list_cursor_is_on_head** (const **gdsl_list_cursor_t** C)

  Check if a cursor is on the head of its list.

- **bool gdsl_list_cursor_is_on_tail** (const **gdsl_list_cursor_t** C)

  Check if a cursor is on the tail of its list.

- **bool gdsl_list_cursor_has_succ** (const **gdsl_list_cursor_t** C)

  Check if a cursor has a successor.

- **bool gdsl_list_cursor_has_pred** (const **gdsl_list_cursor_t** C)

  Check if a cursor has a predecessor.

- void **gdsl_list_cursor_set_content** (**gdsl_list_cursor_t** C, **gdsl_element_t** E)

  Set the content of the cursor.

- **gdsl_element_t gdsl_list_cursor_get_content** (const **gdsl_list_cursor_t** C)

  Get the content of a cursor.

- **gdsl_element_t gdsl_list_cursor_insert_after** (**gdsl_list_cursor_t** C, void *VALUE)

  Insert a new element after a cursor.

- **gdsl_element_t gdsl_list_cursor_insert_before** (**gdsl_list_cursor_t** C, void *VALUE)

  Insert a new element before a cursor.

- **gdsl_element_t gdsl_list_cursor_remove** (**gdsl_list_cursor_t** C)

  Removec the element under a cursor.

- **gdsl_element_t gdsl_list_cursor_remove_after** (**gdsl_list_cursor_t** C)

  Removec the element after a cursor.

- **gdsl_element_t gdsl_list_cursor_remove_before** (**gdsl_list_cursor_t** C)

  Remove the element before a cursor.

- **gdsl_list_cursor_t gdsl_list_cursor_delete** (**gdsl_list_cursor_t** C)

  Delete the element under a cursor.

- **gdsl_list_cursor_t gdsl_list_cursor_delete_after** (**gdsl_list_cursor_t** C)

  Delete the element after a cursor.

- **gdsl_list_cursor_t gdsl_list_cursor_delete_before (gdsl_list_cursor_t C)**

  Delete the element before the cursor of a list.

3.9.1 Typedef Documentation

3.9.1.1 typedef struct _gdsl_list_cursor* gdsl_list_cursor_t

GDSL doubly-linked list cursor type.

This type is voluntary opaque. Variables of this kind could'nt be directly used, but by the functions of this module.

Definition at line 58 of file gdsl_list.h.

3.9.1.2 typedef struct _gdsl_list* gdsl_list_t

GDSL doubly-linked list type.

This type is voluntary opaque. Variables of this kind could'nt be directly used, but by the functions of this module.

Definition at line 50 of file gdsl_list.h.

3.9.2 Function Documentation

3.9.2.1 gdsl_list_t gdsl_list_alloc (const char * *NAME*, gdsl_alloc_func_t *ALLOC_F*, gdsl_free_func_t *FREE_F*)

Create a new list.

Allocate a new list data structure which name is set to a copy of NAME. The function pointers ALLOC_F and FREE_F could be used to respectively, alloc and free elements in the list. These pointers could be set to NULL to use the default ones:

- the default ALLOC_F simply returns its argument
- the default FREE_F does nothing

Note:
Complexity: O(1)

Precondition:
nothing

Parameters:
NAME The name of the new list to create

ALLOC_F Function to alloc element when inserting it in the list

FREE_F Function to free element when removing it from the list

Returns:
the newly allocated list in case of success.
NULL in case of insufficient memory.

See also:
gdsl_list_free()(p. 112)
gdsl_list_flush()(p. 111)

3.9.2.2 gdsl_list_cursor_t gdsl_list_cursor_alloc (const gdsl_list_t *L*)

Create a new list cursor.

Note:
Complexity: O(1)

Precondition:
L must be a valid gdsl_list_t

Parameters:
L The list on wich the cursor is positionned.

Returns:
the newly allocated list cursor in case of success.
NULL in case of insufficient memory.

See also:
gdsl_list_cursor_free()(p. 101)

3.9.2.3 gdsl_list_cursor_t gdsl_list_cursor_delete (gdsl_list_cursor_t *C*)

Delete the element under a cursor.

Remove the element under the cursor C. The removed element is also deallocated using FREE_F passed to **gdsl_list_alloc()**(p. 98).

Complexity: O(1)

Precondition:
C must be a valid gdsl_list_cursor_t

Parameters:
C The cursor to delete the content.

Returns:
the cursor C if the element was removed.
NULL if there is not element to remove.

See also:
gdsl_list_cursor_delete_before()(p. 100)
gdsl_list_cursor_delete_after()(p. 100)

3.9.2.4 gdsl_list_cursor_t gdsl_list_cursor_delete_after (gdsl_list_cursor_t *C*)

Delete the element after a cursor.

Remove the element after the cursor C. The removed element is also deallocated using FREE_F passed to **gdsl_list_alloc()**(p. 98).

Complexity: O(1)

Precondition:
C must be a valid gdsl_list_cursor_t

Parameters:
C The cursor to delete the successor from.

Returns:
the cursor C if the element was removed.
NULL if there is not element to remove.

See also:
gdsl_list_cursor_delete()(p. 99)
gdsl_list_cursor_delete_before()(p. 100)

3.9.2.5 gdsl_list_cursor_t gdsl_list_cursor_delete_before (gdsl_list_cursor_t *C*)

Delete the element before the cursor of a list.

Remove the element before the cursor C. The removed element is also deallocated using FREE_F passed to **gdsl_list_alloc()**(p. 98).

Note:
Complexity: O(1)

Precondition:
C must be a valid gdsl_list_cursor_t

Parameters:
C The cursor to delete the predecessor from.

Returns:
the cursor C if the element was removed.
NULL if there is not element to remove.

See also:
gdsl_list_cursor_delete()(p. 99)
gdsl_list_cursor_delete_after()(p. 100)

3.9.2.6 void gdsl_list_cursor_free (gdsl_list_cursor_t *C*)

Destroy a list cursor.

Note:
Complexity: O(1)

Precondition:
C must be a valid gdsl_list_cursor_t.

Parameters:
C The list cursor to destroy.

See also:
gdsl_list_cursor_alloc()(p. 99)

3.9.2.7 gdsl_element_t gdsl_list_cursor_get_content (const gdsl_list_cursor_t *C*)

Get the content of a cursor.

Note:
Complexity: O(1)

Precondition:
C must be a valid gdsl_list_cursor_t

Parameters:
C The cursor to get the content from.

Returns:
the element contained in the cursor C.

See also:
gdsl_list_cursor_set_content()(p. 108)

3.9.2.8 bool gdsl_list_cursor_has_pred (const gdsl_list_cursor_t *C*)

Check if a cursor has a predecessor.

Note:
Complexity: O(1)

Precondition:
C must be a valid gdsl_list_cursor_t

Parameters:
C The cursor to check

Returns:
TRUE if there exists an element before the cursor C.
FALSE if there is no element before the cursor C.

See also:
gdsl_list_cursor_has_succ()(p. 102)

3.9.2.9 bool gdsl_list_cursor_has_succ (const gdsl_list_cursor_t *C*)

Check if a cursor has a successor.

Note:
Complexity: O(1)

Precondition:
C must be a valid gdsl_list_cursor_t

Parameters:
C The cursor to check

Returns:
TRUE if there exists an element after the cursor C.
FALSE if there is no element after the cursor C.

See also:
gdsl_list_cursor_has_pred()(p. 102)

3.9.2.10 gdsl_element_t gdsl_list_cursor_insert_after (gdsl_list_cursor_t *C*, void * *VALUE*)

Insert a new element after a cursor.

A new element is created using ALLOC_F called on VALUE. ALLOC_F is the pointer passed to **gdsl_list_alloc()**(p. 98). If the returned value is not NULL, then the new element is placed after the cursor C. If C's list is empty, the element is inserted at the head position of C's list.

Note:
Complexity: O(1)

Precondition:
C must be a valid gdsl_list_cursor_t

Parameters:
C The cursor after which the new element must be inserted

VALUE The value used to allocate the new element to insert

Returns:
the newly inserted element in case of success.
NULL in case of failure.

See also:
gdsl_list_cursor_insert_before()(p. 103)
gdsl_list_cursor_remove_after()(p. 107)
gdsl_list_cursor_remove_before()(p. 107)

3.9.2.11 gdsl_element_t gdsl_list_cursor_insert_before (gdsl_list_cursor_t *C*, void * *VALUE*)

Insert a new element before a cursor.

A new element is created using ALLOC_F called on VALUE. ALLOC_F is the pointer passed to **gdsl_list_alloc()**(p. 98). If the returned value is not NULL, then the new element is placed before the cursor C. If C's list is empty, the element is inserted at the head position of C's list.

Note:
Complexity: O(1)

Precondition:
C must be a valid gdsl_list_cursor_t

Parameters:
C The cursor before which the new element must be inserted

VALUE The value used to allocate the new element to insert

Returns:
the newly inserted element in case of success.
NULL in case of failure.

See also:
gdsl_list_cursor_insert_after()(p. 102)
gdsl_list_cursor_remove_after()(p. 107)
gdsl_list_cursor_remove_before()(p. 107)

3.9.2.12 bool gdsl_list_cursor_is_on_head (const gdsl_list_cursor_t C)

Check if a cursor is on the head of its list.

Note:
Complexity: O(1)

Precondition:
C must be a valid gdsl_list_cursor_t

Parameters:
C The cursor to check

Returns:
TRUE if C is on its list's head.
FALSE if C is not on its lits's head.

See also:
gdsl_list_cursor_is_on_tail()(p. 104)

3.9.2.13 bool gdsl_list_cursor_is_on_tail (const gdsl_list_cursor_t C)

Check if a cursor is on the tail of its list.

Note:
Complexity: O(1)

Precondition:
C must be a valid gdsl_list_cursor_t

Parameters:
C The cursor to check

Returns:
TRUE if C is on its lists's tail.
FALSE if C is not on its list's tail.

See also:
gdsl_list_cursor_is_on_head()(p. 104)

3.9.2.14 void gdsl_list_cursor_move_to_head (gdsl_list_cursor_t C)

Put a cursor on the head of its list.

Put the cursor C on the head of C's list. Does nothing if C's list is empty.

Note:
Complexity: O(1)

Precondition:
C must be a valid gdsl_list_cursor_t

Parameters:
C The cursor to use

See also:
gdsl_list_cursor_move_to_tail()(p. 105)

3.9.2.15 gdsl_element_t gdsl_list_cursor_move_to_position (gdsl_list_cursor_t *C*, ulong *POS*)

Place a cursor on a element given by its position.

Search for the POS-th element in the cursor's list L. In case this element exists, the cursor C is positionned on it.

Note:
Complexity: O(|L| / 2)

Precondition:
C must be a valid gdsl_list_cursor_t & POS > 0 & POS <= |L|

Parameters:
C The cursor to put on the POS-th element

POS The position of the element to move on

Returns:
the element at the POS-th position
NULL if POS <= 0 or POS > |L|

See also:
gdsl_list_cursor_move_to_value()(p. 106)

3.9.2.16 void gdsl_list_cursor_move_to_tail (gdsl_list_cursor_t *C*)

Put a cursor on the tail of its list.

Put the cursor C on the tail of C's list. Does nothing if C's list is empty.

Note:
Complexity: O(1)

Precondition:
C must be a valid gdsl_list_cursor_t

Parameters:
C The cursor to use

See also:
gdsl_list_cursor_move_to_head()(p. 104)

3.9.2.17 gdsl_element_t gdsl_list_cursor_move_to_value (gdsl_list_cursor_t *C*, gdsl_compare_func_t *COMP_F*, void * *VALUE*)

Place a cursor on a particular element.

Search a particular element E in the cursor's list L by comparing all list's elements to VALUE, by using COMP_F. If E is found, C is positionned on it.

Note:
Complexity: O(|L| / 2)

Precondition:
C must be a valid gdsl_list_cursor_t & COMP_F != NULL

Parameters:
C The cursor to put on the element E

COMP_F The comparison function to search for E

VALUE The value used to compare list's elements with

Returns:
the first founded element E in case it exists.
NULL in case of element E is not found.

See also:
gdsl_list_cursor_move_to_position()(p. 105)

3.9.2.18 gdsl_element_t gdsl_list_cursor_remove (gdsl_list_cursor_t *C*)

Removec the element under a cursor.

Note:
Complexity: O(1)

Precondition:
C must be a valid gdsl_list_cursor_t

Postcondition:
After this operation, the cursor is positionned on to its successor.

Parameters:
C The cursor to remove the content from.

Returns:
the removed element if it exists.
NULL if there is not element to remove.

See also:
gdsl_list_cursor_insert_after()(p. 102)
gdsl_list_cursor_insert_before()(p. 103)
gdsl_list_cursor_remove()(p. 106)
gdsl_list_cursor_remove_before()(p. 107)

3.9.2.19 gdsl_element_t gdsl_list_cursor_remove_after (gdsl_list_cursor_t C)

Removec the element after a cursor.

Note:
Complexity: O(1)

Precondition:
C must be a valid gdsl_list_cursor_t

Parameters:
C The cursor to remove the successor from.

Returns:
the removed element if it exists.
NULL if there is not element to remove.

See also:
gdsl_list_cursor_insert_after()(p. 102)
gdsl_list_cursor_insert_before()(p. 103)
gdsl_list_cursor_remove()(p. 106)
gdsl_list_cursor_remove_before()(p. 107)

3.9.2.20 gdsl_element_t gdsl_list_cursor_remove_before (gdsl_list_cursor_t C)

Remove the element before a cursor.

Note:
Complexity: O(1)

Precondition:
C must be a valid gdsl_list_cursor_t

Parameters:
C The cursor to remove the predecessor from.

Returns:
the removed element if it exists.
NULL if there is not element to remove.

See also:
gdsl_list_cursor_insert_after()(p. 102)
gdsl_list_cursor_insert_before()(p. 103)
gdsl_list_cursor_remove()(p. 106)
gdsl_list_cursor_remove_after()(p. 107)

3.9.2.21 void gdsl_list_cursor_set_content (gdsl_list_cursor_t *C*, gdsl_element_t *E*)

Set the content of the cursor.

Set C's element to E. The previous element is *NOT* deallocated. If it must be deallocated, **gdsl_list_cursor_get_content()**(p. 101) could be used to get it in order to free it before.

Note:
Complexity: O(1)

Precondition:
C must be a valid gdsl_list_cursor_t

Parameters:
C The cursor in which the content must be modified.
E The value used to modify C's content.

See also:
gdsl_list_cursor_get_content()(p. 101)

3.9.2.22 void gdsl_list_cursor_step_backward (gdsl_list_cursor_t *C*)

Move a cursor one step backward of its list.

Move the cursor C one node backward (from tail to head.) Does nothing if C is already on its list's head.

Note:
Complexity: O(1)

Precondition:
C must be a valid gdsl_list_cursor_t

Parameters:
C The cursor to use

See also:
gdsl_list_cursor_step_forward()(p. 109)

3.9.2.23 void gdsl_list_cursor_step_forward (gdsl_list_cursor_t *C*)

Move a cursor one step forward of its list.

Move the cursor C one node forward (from head to tail). Does nothing if C is already on its list's tail.

Note:
Complexity: O(1)

Precondition:
C must be a valid gdsl_list_cursor_t

Parameters:
C The cursor to use

See also:
gdsl_list_cursor_step_backward()(p. 108)

3.9.2.24 gdsl_list_t gdsl_list_delete (gdsl_list_t *L*, gdsl_compare_func_t *COMP_F*, const void * *VALUE*)

Delete a particular element from a list.

Search into the list L for the first element E equal to VALUE by using COMP_F. If E is found, it is removed from L and deallocated using the FREE_F function passed to **gdsl_list_alloc()**(p. 98).

Note:
Complexity: O(|L| / 2)

Precondition:
L must be a valid gdsl_list_t & COMP_F != NULL

Parameters:
L The list to destroy the element from

COMP_F The comparison function used to find the element to destroy

VALUE The value used to compare the element to destroy with

Returns:
the modified list L if the element is found.
NULL if the element to destroy is not found.

See also:
gdsl_list_alloc()(p. 98)
gdsl_list_destroy_head()
gdsl_list_destroy_tail()

3.9.2.25 gdsl_list_t gdsl_list_delete_head (gdsl_list_t *L*)

Delete the head of a list.

Remove the header element from the list L and deallocates it using the FREE_F function passed to **gdsl_list_alloc()**(p. 98).

Note:
Complexity: O(1)

Precondition:
L must be a valid gdsl_list_t

Parameters:
L The list to destroy the head from

Returns:
the modified list L in case of success.
NULL if L is empty.

See also:
gdsl_list_alloc()(p. 98)
gdsl_list_destroy_tail()
gdsl_list_destroy()

3.9.2.26 gdsl_list_t gdsl_list_delete_tail (gdsl_list_t *L*)

Delete the tail of a list.

Remove the footer element from the list L and deallocates it using the FREE_F function passed to **gdsl_list_alloc()**(p. 98).

Note:
Complexity: O(1)

Precondition:
L must be a valid gdsl_list_t

Parameters:
L The list to destroy the tail from

Returns:
the modified list L in case of success.
NULL if L is empty.

See also:
gdsl_list_alloc()(p. 98)
gdsl_list_destroy_head()
gdsl_list_destroy()

3.9.2.27 void gdsl_list_dump (const gdsl_list_t *L*, gdsl_write_func_t *WRITE_F*, FILE * *OUTPUT_FILE*, void * *USER_DATA*)

Dump the internal structure of a list to a file.

Dump the structure of the list L to OUTPUT_FILE. If WRITE_F != NULL, then uses WRITE_F to write L's elements to OUTPUT_FILE. Additionnal USER_DATA argument could be passed to WRITE_F.

Note:
Complexity: O(|L|)

Precondition:
L must be a valid gdsl_list_t & OUTPUT_FILE != NULL

Parameters:
L The list to write.
WRITE_F The write function.
OUTPUT_FILE The file where to write L's elements.
USER_DATA User's datas passed to WRITE_F.

See also:
gdsl_list_write()(p. 122)
gdsl_list_write_xml()(p. 122)

3.9.2.28 void gdsl_list_flush (gdsl_list_t *L*)

Flush a list.

Destroy all the elements of the list L by calling L's FREE_F function passed to **gdsl_list_alloc()**(p. 98). L is not deallocated itself and L's name is not modified.

Note:
Complexity: O(|L|)

Precondition:
L must be a valid gdsl_list_t

Parameters:
L The list to flush

See also:
gdsl_list_alloc()(p. 98)
gdsl_list_free()(p. 112)

3.9.2.29 void gdsl_list_free (gdsl_list_t *L*)

Destroy a list.

Flush and destroy the list L. All the elements of L are freed using L's FREE_F function passed to **gdsl_list_alloc()**(p. 98).

Note:
Complexity: O(|L|)

Precondition:
L must be a valid gdsl_list_t

Parameters:
L The list to destroy

See also:
gdsl_list_alloc()(p. 98)
gdsl_list_flush()(p. 111)

3.9.2.30 gdsl_element_t gdsl_list_get_head (const gdsl_list_t *L*)

Get the head of a list.

Note:
Complexity: O(1)

Precondition:
L must be a valid gdsl_list_t

Parameters:
L The list to get the head from

Returns:
the element at L's head position if L is not empty. The returned element is not removed from L.
NULL if the list L is empty.

See also:
gdsl_list_get_tail()(p. 113)

3.9.2.31 const char* gdsl_list_get_name (const gdsl_list_t *L*)

Get the name of a list.

Note:
Complexity: O(1)

Precondition:
L must be a valid gdsl_list_t

Postcondition:
The returned string MUST NOT be freed.

Parameters:
L The list to get the name from

Returns:
the name of the list L.

See also:
gdsl_list_set_name()(p. 121)

3.9.2.32 ulong gdsl_list_get_size (const gdsl_list_t *L*)

Get the size of a list.

Note:
Complexity: O(1)

Precondition:
L must be a valid gdsl_list_t

Parameters:
L The list to get the size from

Returns:
the number of elements of the list L (noted |L|).

3.9.2.33 gdsl_element_t gdsl_list_get_tail (const gdsl_list_t *L*)

Get the tail of a list.

Note:
Complexity: O(1)

Precondition:
L must be a valid gdsl_list_t

Parameters:
L The list to get the tail from

Returns:
the element at L's tail position if L is not empty. The returned element is not removed from L.
NULL if L is empty.

See also:
gdsl_list_get_head()(p. 112)

3.9.2.34 gdsl_element_t gdsl_list_insert_head (gdsl_list_t *L*, void * *VALUE*)

Insert an element at the head of a list.

Allocate a new element E by calling L's ALLOC_F function on VALUE. ALLOC_F is the function pointer passed to **gdsl_list_alloc()**(p. 98). The new element E is then inserted at the header position of the list L.

Note:
Complexity: O(1)

Precondition:
L must be a valid gdsl_list_t

Parameters:
L The list to insert into

VALUE The value used to make the new element to insert into L

Returns:
the inserted element E in case of success.
NULL in case of failure.

See also:
gdsl_list_insert_tail()(p. 114)
gdsl_list_remove_head()(p. 117)
gdsl_list_remove_tail()(p. 118)
gdsl_list_remove()(p. 117)

3.9.2.35 gdsl_element_t gdsl_list_insert_tail (gdsl_list_t *L*, void * *VALUE*)

Insert an element at the tail of a list.

Allocate a new element E by calling L's ALLOC_F function on VALUE. ALLOC_F is the function pointer passed to **gdsl_list_alloc()**(p. 98). The new element E is then inserted at the footer position of the list L.

Note:
Complexity: O(1)

Precondition:
L must be a valid gdsl_list_t

Parameters:
L The list to insert into

VALUE The value used to make the new element to insert into L

Returns:
the inserted element E in case of success.
NULL in case of failure.

See also:
gdsl_list_insert_head()(p. 114)
gdsl_list_remove_head()(p. 117)
gdsl_list_remove_tail()(p. 118)
gdsl_list_remove()(p. 117)

3.9.2.36 bool gdsl_list_is_empty (const gdsl_list_t *L*)

Check if a list is empty.

Note:
Complexity: O(1)

Precondition:
L must be a valid gdsl_list_t

Parameters:
L The list to check

Returns:
TRUE if the list L is empty.
FALSE if the list L is not empty.

3.9.2.37 gdsl_element_t gdsl_list_map_backward (const gdsl_list_t *L*, gdsl_map_func_t *MAP_F*, void * *USER_DATA*)

Parse a list from tail to head.

Parse all elements of the list L from tail to head. The MAP_F function is called on each L's element with USER_DATA argument. If MAP_F returns GDSL_MAP_STOP then **gdsl_list_map_backward()**(p. 115) stops and returns its last examinated element.

Note:
Complexity: O(|L|)

Precondition:
L must be a valid gdsl_list_t & MAP_F != NULL

Parameters:
L The list to parse

MAP_F The map function to apply on each L's element

USER_DATA User's datas passed to MAP_F

Returns:
the first element for which MAP_F returns GDSL_MAP_STOP.
NULL when the parsing is done.

See also:
gdsl_list_map_forward()(p. 116)

3.9.2.38 gdsl_element_t gdsl_list_map_forward (const gdsl_list_t *L*, gdsl_map_func_t *MAP_F*, void * *USER_DATA*)

Parse a list from head to tail.

Parse all elements of the list L from head to tail. The MAP_F function is called on each L's element with USER_DATA argument. If MAP_F returns GDSL_MAP_STOP, then **gdsl_list_map_forward()**(p. 116) stops and returns its last examinated element.

Note:
Complexity: O(|L|)

Precondition:
L must be a valid gdsl_list_t & MAP_F != NULL

Parameters:
L The list to parse

MAP_F The map function to apply on each L's element

USER_DATA User's datas passed to MAP_F

Returns:
the first element for which MAP_F returns GDSL_MAP_STOP.
NULL when the parsing is done.

See also:
gdsl_list_map_backward()(p. 115)

3.9.2.39 gdsl_element_t gdsl_list_remove (gdsl_list_t *L*, gdsl_compare_func_t *COMP_F*, const void * *VALUE*)

Remove a particular element from a list.

Search into the list L for the first element E equal to VALUE by using COMP_F. If E is found, it is removed from L and then returned.

Note:
Complexity: O(|L| / 2)

Precondition:
L must be a valid gdsl_list_t & COMP_F != NULL

Parameters:
L The list to remove the element from

COMP_F The comparison function used to find the element to remove

VALUE The value used to compare the element to remove with

Returns:
the founded element E if it was found.
NULL in case the searched element E was not found.

See also:
gdsl_list_insert_head()(p. 114)
gdsl_list_insert_tail()(p. 114)
gdsl_list_remove_head()(p. 117)
gdsl_list_remove_tail()(p. 118)

3.9.2.40 gdsl_element_t gdsl_list_remove_head (gdsl_list_t *L*)

Remove the head of a list.

Remove the element at the head of the list L.

Note:
Complexity: O(1)

Precondition:
L must be a valid gdsl_list_t

Parameters:
L The list to remove the head from

Returns:
the removed element in case of success.
NULL in case of L is empty.

See also:
gdsl_list_insert_head()(p. 114)
gdsl_list_insert_tail()(p. 114)
gdsl_list_remove_tail()(p. 118)
gdsl_list_remove()(p. 117)

3.9.2.41 gdsl_element_t gdsl_list_remove_tail (gdsl_list_t *L*)

Remove the tail of a list.

Remove the element at the tail of the list L.

Note:
Complexity: O(1)

Precondition:
L must be a valid gdsl_list_t

Parameters:
L The list to remove the tail from

Returns:
the removed element in case of success.
NULL in case of L is empty.

See also:
gdsl_list_insert_head()(p. 114)
gdsl_list_insert_tail()(p. 114)
gdsl_list_remove_head()(p. 117)
gdsl_list_remove()(p. 117)

3.9.2.42 gdsl_element_t gdsl_list_search (const gdsl_list_t *L*, gdsl_compare_func_t *COMP_F*, const void * *VALUE*)

Search for a particular element into a list.

Search the first element E equal to VALUE in the list L, by using COMP_F to compare all L's element with.

Note:
Complexity: O(|L| / 2)

Precondition:
L must be a valid gdsl_list_t & COMP_F != NULL

Parameters:
L The list to search the element in

COMP_F The comparison function used to compare L's element with VALUE

VALUE The value to compare L's elemenst with

Returns:
the first founded element E in case of success.
NULL in case the searched element E was not found.

See also:
gdsl_list_search_by_position()(p. 119)
gdsl_list_search_max()(p. 119)
gdsl_list_search_min()(p. 120)

3.9.2.43 gdsl_element_t gdsl_list_search_by_position (const gdsl_list_t *L*, ulong *POS*)

Search for an element by its position in a list.

Note:
Complexity: O(|L| / 2)

Precondition:
L must be a valid gdsl_list_t & POS > 0 & POS <= |L|

Parameters:
L The list to search the element in

POS The position where is the element to search

Returns:
the element at the POS-th position in the list L.
NULL if POS > |L| or POS <= 0.

See also:
gdsl_list_search()(p. 118)
gdsl_list_search_max()(p. 119)
gdsl_list_search_min()(p. 120)

3.9.2.44 gdsl_element_t gdsl_list_search_max (const gdsl_list_t *L*, gdsl_compare_func_t *COMP_F*)

Search for the greatest element of a list.

Search the greatest element of the list L, by using COMP_F to compare L's elements with.

Note:
Complexity: O(|L|)

Precondition:
L must be a valid gdsl_list_t & COMP_F != NULL

Parameters:
L The list to search the element in

COMP_F The comparison function to use to compare L's element with

Returns:
the highest element of L, by using COMP_F function.
NULL if L is empty.

See also:
gdsl_list_search()(p. 118)
gdsl_list_search_by_position()(p. 119)
gdsl_list_search_min()(p. 120)

3.9.2.45 gdsl_element_t gdsl_list_search_min (const gdsl_list_t *L*, gdsl_compare_func_t *COMP_F*)

Search for the lowest element of a list.

Search the lowest element of the list L, by using COMP_F to compare L's elements with.

Note:
Complexity: O(|L|)

Precondition:
L must be a valid gdsl_list_t & COMP_F != NULL

Parameters:
L The list to search the element in

COMP_F The comparison function to use to compare L's element with

Returns:
the lowest element of L, by using COMP_F function.
NULL if L is empty.

See also:
gdsl_list_search()(p. 118)
gdsl_list_search_by_position()(p. 119)
gdsl_list_search_max()(p. 119)

3.9.2.46 gdsl_list_t gdsl_list_set_name (gdsl_list_t *L*, const char * *NEW_NAME*)

Set the name of a list.

Changes the previous name of the list L to a copy of NEW_NAME.

Note:
Complexity: O(1)

Precondition:
L must be a valid gdsl_list_t

Parameters:
L The list to change the name

NEW_NAME The new name of L

Returns:
the modified list in case of success.
NULL in case of failure.

See also:
gdsl_list_get_name()(p. 113)

3.9.2.47 gdsl_list_t gdsl_list_sort (gdsl_list_t *L*, gdsl_compare_func_t *COMP_F*, gdsl_element_t *MAX*)

Sort a list.

Sort the list L using COMP_F to order L's elements.

Note:
Complexity: O(|L| * log(|L|))
VERY IMPORTANT: L must *NOT* contains an element >= MAX.

Precondition:
L must be a valid gdsl_list_t & COMP_F != NULL & L must not contains elements that are equals & MAX must be higger than *ALL* L's elements

Parameters:
L The list to sort

COMP_F The comparison function used to order L's elements

MAX An element greather than all other L's ones

Returns:
the sorted list L.

3.9.2.48 void gdsl_list_write (const gdsl_list_t *L*, gdsl_write_func_t *WRITE_F*, FILE * *OUTPUT_FILE*, void * *USER_DATA*)

Write all the elements of a list to a file.

Write the elements of the list L to OUTPUT_FILE, using WRITE_F function. Additionnal USER_DATA argument could be passed to WRITE_F.

Note:
Complexity: O(|L|)

Precondition:
L must be a valid gdsl_list_t & OUTPUT_FILE != NULL & WRITE_F != NULL

Parameters:
L The list to write.

WRITE_F The write function.

OUTPUT_FILE The file where to write L's elements.

USER_DATA User's datas passed to WRITE_F.

See also:
gdsl_list_write_xml()(p. 122)
gdsl_list_dump()(p. 111)

3.9.2.49 void gdsl_list_write_xml (const gdsl_list_t *L*, gdsl_write_func_t *WRITE_F*, FILE * *OUTPUT_FILE*, void * *USER_DATA*)

Write the content of a list to a file into XML.

Write the elements of the list L to OUTPUT_FILE, into XML language. If WRITE_F != NULL, then uses WRITE_F to write L's elements to OUTPUT_FILE. Additionnal USER_DATA argument could be passed to WRITE_F.

Note:
Complexity: O(|L|)

Precondition:
L must be a valid gdsl_list_t & OUTPUT_FILE != NULL

Parameters:
L The list to write.

WRITE_F The write function.

OUTPUT_FILE The file where to write L's elements.

USER_DATA User's datas passed to WRITE_F.

See also:
gdsl_list_write()(p. 122)
gdsl_list_dump()(p. 111)

3.10 Various macros module

Defines

- #define **GDSL_MAX**(X, Y) (X>Y?X:Y)

  Give the greatest number of two numbers.

- #define **GDSL_MIN**(X, Y) (X>Y?Y:X)

  Give the lowest number of two numbers.

3.10.1 Define Documentation

3.10.1.1 #define GDSL_MAX(X, Y) (X>Y?X:Y)

Give the greatest number of two numbers.

Note:
Complexity: O(1)

Precondition:
X & Y must be basic scalar C types

Parameters:
X First scalar variable

Y Second scalar variable

Returns:
X if X is greather than Y.
Y if Y is greather than X.

See also:
GDSL_MIN()(p. 124)

Definition at line 55 of file gdsl_macros.h.

3.10.1.2 #define GDSL_MIN(X, Y) (X>Y?Y:X)

Give the lowest number of two numbers.

Note:
Complexity: O(1)

Precondition:
X & Y must be basic scalar C types

Parameters:

X First scalar variable

Y Second scalar variable

Returns:

Y if Y is lower than X.
X if X is lower than Y.

See also:

GDSL_MAX()(p. 124)

Definition at line 72 of file gdsl_macros.h.

3.11 Permutation manipulation module

Typedefs

- typedef gdsl_perm * **gdsl_perm_t**

  GDSL permutation type.

- typedef void(* **gdsl_perm_write_func_t**)(**ulong** E, FILE *OUTPUT_FILE, **gdsl_perm_position_t** POSITION, void *USER_DATA)

  GDSL permutation write function type.

Enumerations

- enum **gdsl_perm_position_t** { **GDSL_PERM_POSITION_FIRST** = 1, **GDSL_PERM_POSITION_LAST** = 2 }

  This type is for gdsl_perm_write_func_t.

Functions

- **gdsl_perm_t gdsl_perm_alloc** (const char *NAME, const **ulong** N)

  Create a new permutation.

- void **gdsl_perm_free** (**gdsl_perm_t** P)

  Destroy a permutation.

- **gdsl_perm_t gdsl_perm_copy** (const **gdsl_perm_t** P)

  Copy a permutation.

- const char * **gdsl_perm_get_name** (const **gdsl_perm_t** P)

  Get the name of a permutation.

- **ulong gdsl_perm_get_size** (const **gdsl_perm_t** P)

  Get the size of a permutation.

- **ulong gdsl_perm_get_element** (const **gdsl_perm_t** P, const **ulong** INDIX)

  Get the (INDIX+1)-th element from a permutation.

- **ulong** * **gdsl_perm_get_elements_array** (const **gdsl_perm_t** P)

  Get the array elements of a permutation.

- **ulong gdsl_perm_linear_inversions_count** (const **gdsl_perm_t** P)

  Count the inversions number into a linear permutation.

- **ulong gdsl_perm_linear_cycles_count** (const **gdsl_perm_t** P)

  Count the cycles number into a linear permutation.

- **ulong gdsl_perm_canonical_cycles_count** (const **gdsl_perm_t** P)

  Count the cycles number into a canonical permutation.

- **gdsl_perm_t gdsl_perm_set_name** (**gdsl_perm_t** P, const char *NEW_NAME)

  Set the name of a permutation.

- **gdsl_perm_t gdsl_perm_linear_next** (**gdsl_perm_t** P)

  Get the next permutation from a linear permutation.

- **gdsl_perm_t gdsl_perm_linear_prev** (**gdsl_perm_t** P)

  Get the previous permutation from a linear permutation.

- **gdsl_perm_t gdsl_perm_set_elements_array** (**gdsl_perm_t** P, const **ulong** *ARRAY)

  Initialize a permutation with an array of values.

- **gdsl_perm_t gdsl_perm_multiply** (**gdsl_perm_t** RESULT, const **gdsl_perm_t** ALPHA, const **gdsl_perm_t** BETA)

  Multiply two permutations.

- **gdsl_perm_t gdsl_perm_linear_to_canonical** (**gdsl_perm_t** Q, const **gdsl_perm_t** P)

  Convert a linear permutation to its canonical form.

- **gdsl_perm_t gdsl_perm_canonical_to_linear** (**gdsl_perm_t** Q, const **gdsl_perm_t** P)

  Convert a canonical permutation to its linear form.

- **gdsl_perm_t gdsl_perm_inverse** (**gdsl_perm_t** P)

  Inverse in place a permutation.

- **gdsl_perm_t gdsl_perm_reverse** (**gdsl_perm_t** P)

  Reverse in place a permutation.

- **gdsl_perm_t gdsl_perm_randomize** (**gdsl_perm_t** P)

  Randomize a permutation.

- **gdsl_element_t * gdsl_perm_apply_on_array** (**gdsl_element_t** *V, const **gdsl_perm_t** P)

  Apply a permutation on to a vector.

- void **gdsl_perm_write** (const **gdsl_perm_t** P, const **gdsl_perm_write_func_t** WRITE_F, FILE *OUTPUT_FILE, void *USER_DATA)

  Write the elements of a permutation to a file.

- void **gdsl_perm_write_xml** (const **gdsl_perm_t** P, const **gdsl_write_func_t** WRITE_F, FILE *OUTPUT_FILE, void *USER_DATA)

  Write the elements of a permutation to a file into XML.

- void **gdsl_perm_dump** (const **gdsl_perm_t** P, const **gdsl_write_func_t** WRITE_F, FILE *OUTPUT_FILE, void *USER_DATA)

  Dump the internal structure of a permutation to a file.

3.11.1 Typedef Documentation

3.11.1.1 typedef struct gdsl_perm* gdsl_perm_t

GDSL permutation type.

This type is voluntary opaque. Variables of this kind could'nt be directly used, but by the functions of this module.

Definition at line 49 of file gdsl_perm.h.

3.11.1.2 typedef void(* gdsl_perm_write_func_t)(ulong E, FILE* OUTPUT_FILE, gdsl_perm_position_t POSITION, void* USER_DATA)

GDSL permutation write function type.

Parameters:

E The permutation element to write

OUTPUT_FILE The file where to write E

POSITION is an or-ed combination of gdsl_perm_position_t values to indicate where E is located into the gdsl_perm_t mapped.

USER_DATA User's datas

Definition at line 73 of file gdsl_perm.h.

3.11.2 Enumeration Type Documentation

3.11.2.1 enum gdsl_perm_position_t

This type is for gdsl_perm_write_func_t.

Enumeration values:

GDSL_PERM_POSITION_FIRST When element is at first position

GDSL_PERM_POSITION_LAST When element is at last position

Definition at line 54 of file gdsl_perm.h.

3.11.3 Function Documentation

3.11.3.1 gdsl_perm_t gdsl_perm_alloc (const char * *NAME*, const ulong *N*)

Create a new permutation.

Allocate a new permutation data structure of size N wich name is set to a copy of NAME.

Note:

Complexity: O(N)

Precondition:

$N > 0$

Parameters:

N The number of elements of the permutation to create.

NAME The name of the new permutation to create

Returns:

the newly allocated identity permutation in its linear form in case of success. NULL in case of insufficient memory.

See also:

gdsl_perm_free()(p. 132)
gdsl_perm_copy()(p. 131)

3.11.3.2 gdsl_element_t* gdsl_perm_apply_on_array (gdsl_element_t * *V*, const gdsl_perm_t *P*)

Apply a permutation on to a vector.

Note:
Complexity: O(|P|)

Precondition:
P must be a valid gdsl_perm_t & |P| == |V|

Parameters:
V The vector/array to reorder according to P

P The permutation to use to reorder V

Returns:
the reordered array V according to the permutation P in case of success. NULL in case of insufficient memory.

3.11.3.3 ulong gdsl_perm_canonical_cycles_count (const gdsl_perm_t *P*)

Count the cycles number into a canonical permutation.

Note:
Complexity: O(|P|)

Precondition:
P must be a valid canonical gdsl_perm_t

Parameters:
P The canonical permutation to use.

Returns:
the number of cycles into the canonical permutation P.

See also:
gdsl_perm_linear_cycles_count()(p. 135)

3.11.3.4 gdsl_perm_t gdsl_perm_canonical_to_linear (gdsl_perm_t *Q*, const gdsl_perm_t *P*)

Convert a canonical permutation to its linear form.

Convert the canonical permutation P to its linear form. The resulted linear permutation is placed into Q without modifying P.

Note:
Complexity: O(|P|)

Precondition:
P & Q must be valids gdsl_perm_t & |P| == |Q| & P != Q

Parameters:

Q The linear form of P

P The canonical permutation used to compute its linear form into Q

Returns:

the linear form Q of the permutation P.

See also:

gdsl_perm_linear_to_canonical()(p. 136)

3.11.3.5 gdsl_perm_t gdsl_perm_copy (const gdsl_perm_t *P*)

Copy a permutation.

Create and return a copy of the permutation P.

Note:

Complexity: O(|P|)

Precondition:

P must be a valid gdsl_perm_t.

Postcondition:

The returned permutation must be deallocated with gdsl_perm_free.

Parameters:

P The permutation to copy.

Returns:

a copy of P in case of success.
NULL in case of insufficient memory.

See also:

gdsl_perm_alloc(p. 129)
gdsl_perm_free(p. 132)

3.11.3.6 void gdsl_perm_dump (const gdsl_perm_t *P*, const gdsl_write_func_t *WRITE_F*, FILE * *OUTPUT_FILE*, void * *USER_DATA*)

Dump the internal structure of a permutation to a file.

Dump the structure of the permutation P to OUTPUT_FILE. If WRITE_F != NULL, then uses WRITE_F function to write P's elements to OUTPUT_FILE. Additionnal USER_DATA argument could be passed to WRITE_F.

Note:

Complexity: O(|P|)

Precondition:
P must be a valid gdsl_perm_t & OUTPUT_FILE != NULL

Parameters:
P The permutation to dump.

WRITE_F The write function.

OUTPUT_FILE The file where to write P's elements.

USER_DATA User's datas passed to WRITE_F.

See also:
gdsl_perm_write()(p. 139)
gdsl_perm_write_xml()(p. 140)

3.11.3.7 void gdsl_perm_free (gdsl_perm_t *P*)

Destroy a permutation.

Deallocate the permutation P.

Note:
Complexity: O(|P|)

Precondition:
P must be a valid gdsl_perm_t

Parameters:
P The permutation to destroy

See also:
gdsl_perm_alloc()(p. 129)
gdsl_perm_copy()(p. 131)

3.11.3.8 ulong gdsl_perm_get_element (const gdsl_perm_t *P*, const ulong *INDIX*)

Get the (INDIX+1)-th element from a permutation.

Note:
Complexity: O(1)

Precondition:
P must be a valid gdsl_perm_t & <= 0 INDIX < |P|

Parameters:
P The permutation to use.

INDIX The indix of the value to get.

Returns:
the value at the INDIX-th position in the permutation P.

See also:
gdsl_perm_get_size()(p. 134)
gdsl_perm_get_elements_array()(p. 133)

3.11.3.9 ulong* gdsl_perm_get_elements_array (const gdsl_perm_t *P*)

Get the array elements of a permutation.

Note:
Complexity: O(1)

Precondition:
P must be a valid gdsl_perm_t

Parameters:
P The permutation to get datas from.

Returns:
the values array of the permutation P.

See also:
gdsl_perm_get_element()(p. 132)
gdsl_perm_set_elements_array()(p. 138)

3.11.3.10 const char* gdsl_perm_get_name (const gdsl_perm_t *P*)

Get the name of a permutation.

Note:
Complexity: O(1)

Precondition:
P must be a valid gdsl_perm_t

Postcondition:
The returned string MUST NOT be freed.

Parameters:
P The permutation to get the name from

Returns:
the name of the permutation P.

See also:
gdsl_perm_set_name()(p. 139)

3.11.3.11 ulong gdsl_perm_get_size (const gdsl_perm_t *P*)

Get the size of a permutation.

Note:
Complexity: O(1)

Precondition:
P must be a valid gdsl_perm_t

Parameters:
P The permutation to get the size from.

Returns:
the number of elements of P (noted |P|).

See also:
gdsl_perm_get_element()(p. 132)
gdsl_perm_get_elements_array()(p. 133)

3.11.3.12 gdsl_perm_t gdsl_perm_inverse (gdsl_perm_t *P*)

Inverse in place a permutation.

Note:
Complexity: O(|P|)

Precondition:
P must be a valid gdsl_perm_t

Parameters:
P The permutation to invert

Returns:
the inverse permutation of P in case of success.
NULL in case of insufficient memory.

See also:
gdsl_perm_reverse()(p. 138)

3.11.3.13 ulong gdsl_perm_linear_cycles_count (const gdsl_perm_t *P*)

Count the cycles number into a linear permutation.

Note:
Complexity: O(|P|)

Precondition:
P must be a valid linear gdsl_perm_t

Parameters:
P The linear permutation to use.

Returns:
the number of cycles into the linear permutation P.

See also:
gdsl_perm_canonical_cycles_count()(p. 130)

3.11.3.14 ulong gdsl_perm_linear_inversions_count (const gdsl_perm_t *P*)

Count the inversions number into a linear permutation.

Note:
Complexity: O(|P|)

Precondition:
P must be a valid linear gdsl_perm_t

Parameters:
P The linear permutation to use.

Returns:
the number of inversions into the linear permutation P.

3.11.3.15 gdsl_perm_t gdsl_perm_linear_next (gdsl_perm_t *P*)

Get the next permutation from a linear permutation.

The permutation P is modified to become the next permutation after P.

Note:
Complexity: O(|P|)

Precondition:
P must be a valid linear gdsl_perm_t & |P| > 1

Parameters:
P The linear permutation to modify

Returns:
the next permutation after the permutation P.
NULL if P is already the last permutation.

See also:
gdsl_perm_linear_prev()(p. 136)

3.11.3.16 gdsl_perm_t gdsl_perm_linear_prev (gdsl_perm_t *P*)

Get the previous permutation from a linear permutation.

The permutation P is modified to become the previous permutation before P.

Note:
Complexity: O(|P|)

Precondition:
P must be a valid linear gdsl_perm_t & |P| >= 2

Parameters:
P The linear permutation to modify

Returns:
the previous permutation before the permutation P.
NULL if P is already the first permutation.

See also:
gdsl_perm_linear_next()(p. 135)

3.11.3.17 gdsl_perm_t gdsl_perm_linear_to_canonical (gdsl_perm_t *Q*, const gdsl_perm_t *P*)

Convert a linear permutation to its canonical form.

Convert the linear permutation P to its canonical form. The resulted canonical permutation is placed into Q without modifying P.

Note:
Complexity: O(|P|)

Precondition:
P & Q must be valids gdsl_perm_t & |P| == |Q| & P != Q

Parameters:

Q The canonical form of P

P The linear permutation used to compute its canonical form into Q

Returns:

the canonical form Q of the permutation P.

See also:

gdsl_perm_canonical_to_linear()(p. 130)

3.11.3.18 gdsl_perm_t gdsl_perm_multiply (gdsl_perm_t *RESULT*, const gdsl_perm_t *ALPHA*, const gdsl_perm_t *BETA*)

Multiply two permutations.

Compute the product of the permutations ALPHA x BETA and puts the result in RESULT without modifying ALPHA and BETA.

Note:

Complexity: O(|RESULT|)

Precondition:

RESULT, ALPHA and BETA must be valids gdsl_perm_t & |RESULT| == |ALPHA| == |BETA|

Parameters:

RESULT The result of the product ALPHA x BETA

ALPHA The first permutation used in the product

BETA The second permutation used in the product

Returns:

RESULT, the result of the multiplication of the permutations A and B.

3.11.3.19 gdsl_perm_t gdsl_perm_randomize (gdsl_perm_t *P*)

Randomize a permutation.

The permutation P is randomized in an efficient way, using inversions array.

Note:

Complexity: O(|P|)

Precondition:

P must be a valid gdsl_perm_t

Parameters:
P The permutation to randomize

Returns:
the mirror image ~P of the permutation of P in case of success.
NULL in case of insufficient memory.

3.11.3.20 gdsl_perm_t gdsl_perm_reverse (gdsl_perm_t *P*)

Reverse in place a permutation.

Note:
Complexity: O(|P| / 2)

Precondition:
P must be a valid gdsl_perm_t

Parameters:
P The permutation to reverse

Returns:
the mirror image of the permutation P

See also:
gdsl_perm_inverse()(p. 134)

3.11.3.21 gdsl_perm_t gdsl_perm_set_elements_array (gdsl_perm_t *P*, const ulong * *ARRAY*)

Initialize a permutation with an array of values.

Initialize the permutation P with the values contained in the array of values ARRAY. If ARRAY does not design a permutation, then P is left unchanged.

Note:
Complexity: O(|P|)

Precondition:
P must be a valid gdsl_perm_t & V != NULL & |V| == |P|

Parameters:
P The permutation to initialize
ARRAY The array of values to initialize P

Returns:
the modified permutation in case of success.
NULL in case V does not design a valid permutation.

See also:
gdsl_perm_get_elements_array()(p. 133)

3.11.3.22 gdsl_perm_t gdsl_perm_set_name (gdsl_perm_t *P*, const char * *NEW_NAME*)

Set the name of a permutation.

Change the previous name of the permutation P to a copy of NEW_NAME.

Note:
Complexity: O(1)

Precondition:
P must be a valid gdsl_perm_t

Parameters:
P The permutation to change the name
NEW_NAME The new name of P

Returns:
the modified permutation in case of success.
NULL in case of insufficient memory.

See also:
gdsl_perm_get_name()(p. 133)

3.11.3.23 void gdsl_perm_write (const gdsl_perm_t *P*, const gdsl_perm_write_func_t *WRITE_F*, FILE * *OUTPUT_FILE*, void * *USER_DATA*)

Write the elements of a permutation to a file.

Write the elements of the permuation P to OUTPUT_FILE, using WRITE_F function. Additionnal USER_DATA argument could be passed to WRITE_F.

Note:
Complexity: O(|P|)

Precondition:
P must be a valid gdsl_perm_t & WRITE_F != NULL & OUTPUT_FILE != NULL

Parameters:
P The permutation to write.
WRITE_F The write function.
OUTPUT_FILE The file where to write P's elements.
USER_DATA User's datas passed to WRITE_F.

See also:
gdsl_perm_write_xml()(p. 140)
gdsl_perm_dump()(p. 131)

3.11.3.24 void gdsl_perm_write_xml (const gdsl_perm_t *P*, const gdsl_write_func_t *WRITE_F*, FILE * *OUTPUT_FILE*, void * *USER_DATA*)

Write the elements of a permutation to a file into XML.

Write the elements of the permutation P to OUTPUT_FILE, into XML language. If WRITE_F != NULL, then uses WRITE_F function to write P's elements to OUTPUT_FILE. Additionnal USER_DATA argument could be passed to WRITE_F.

Note:
Complexity: O(|P|)

Precondition:
P must be a valid gdsl_perm_t & OUTPUT_FILE != NULL

Parameters:
P The permutation to write.

WRITE_F The write function.

OUTPUT_FILE The file where to write P's elements.

USER_DATA User's datas passed to WRITE_F.

See also:
gdsl_perm_write()(p. 139)
gdsl_perm_dump()(p. 131)

3.12 Queue manipulation module

Typedefs

- typedef _gdsl_queue * **gdsl_queue_t**

  GDSL queue type.

Functions

- **gdsl_queue_t gdsl_queue_alloc** (const char *NAME, **gdsl_alloc_func_t** ALLOC_F, **gdsl_free_func_t** FREE_F)

  Create a new queue.

- void **gdsl_queue_free** (**gdsl_queue_t** Q)

  Destroy a queue.

- void **gdsl_queue_flush** (**gdsl_queue_t** Q)

  Flush a queue.

- const char * **gdsl_queue_get_name** (const **gdsl_queue_t** Q)

  Getsthe name of a queue.

- **ulong gdsl_queue_get_size** (const **gdsl_queue_t** Q)

  Get the size of a queue.

- **bool gdsl_queue_is_empty** (const **gdsl_queue_t** Q)

  Check if a queue is empty.

- **gdsl_element_t gdsl_queue_get_head** (const **gdsl_queue_t** Q)

  Get the head of a queue.

- **gdsl_element_t gdsl_queue_get_tail** (const **gdsl_queue_t** Q)

  Get the tail of a queue.

- **gdsl_queue_t gdsl_queue_set_name** (**gdsl_queue_t** Q, const char *NEW_NAME)

  Set the name of a queue.

- **gdsl_element_t gdsl_queue_insert** (**gdsl_queue_t** Q, void *VALUE)

  Insert an element in a queue (PUT).

- **gdsl_element_t gdsl_queue_remove** (**gdsl_queue_t** Q)

  Remove an element from a queue (GET).

- **gdsl_element_t gdsl_queue_search** (const **gdsl_queue_t** Q, **gdsl_compare_func_t** COMP_F, void *VALUE)

  Search for a particular element in a queue.

- **gdsl_element_t gdsl_queue_search_by_position** (const **gdsl_queue_t** Q, **ulong** POS)

  Search for an element by its position in a queue.

- **gdsl_element_t gdsl_queue_map_forward** (const **gdsl_queue_t** Q, **gdsl_map_func_t** MAP_F, void *USER_DATA)

  Parse a queue from head to tail.

- **gdsl_element_t gdsl_queue_map_backward** (const **gdsl_queue_t** Q, **gdsl_map_func_t** MAP_F, void *USER_DATA)

  Parse a queue from tail to head.

- void **gdsl_queue_write** (const **gdsl_queue_t** Q, **gdsl_write_func_t** WRITE_F, FILE *OUTPUT_FILE, void *USER_DATA)

  Write all the elements of a queue to a file.

- void **gdsl_queue_write_xml** (const **gdsl_queue_t** Q, **gdsl_write_func_t** WRITE_F, FILE *OUTPUT_FILE, void *USER_DATA)

  Write the content of a queue to a file into XML.

- void **gdsl_queue_dump** (const **gdsl_queue_t** Q, **gdsl_write_func_t** WRITE_F, FILE *OUTPUT_FILE, void *USER_DATA)

  Dump the internal structure of a queue to a file.

3.12.1 Typedef Documentation

3.12.1.1 typedef struct _gdsl_queue* gdsl_queue_t

GDSL queue type.

This type is voluntary opaque. Variables of this kind could'nt be directly used, but by the functions of this module.

Definition at line 53 of file gdsl_queue.h.

3.12.2 Function Documentation

3.12.2.1 gdsl_queue_t gdsl_queue_alloc (const char * *NAME*, gdsl_alloc_func_t *ALLOC_F*, gdsl_free_func_t *FREE_F*)

Create a new queue.

Allocate a new queue data structure which name is set to a copy of NAME. The functions pointers ALLOC_F and FREE_F could be used to respectively, alloc and free elements in the queue. These pointers could be set to NULL to use the default ones:

- the default ALLOC_F simply returns its argument
- the default FREE_F does nothing

Note:
Complexity: O(1)

Precondition:
nothing.

Parameters:
NAME The name of the new queue to create

ALLOC_F Function to alloc element when inserting it in a queue

FREE_F Function to free element when deleting it from a queue

Returns:
the newly allocated queue in case of success.
NULL in case of insufficient memory.

See also:
gdsl_queue_free()(p. 144)
gdsl_queue_flush()(p. 144)

3.12.2.2 void gdsl_queue_dump (const gdsl_queue_t *Q*, gdsl_write_func_t *WRITE_F*, FILE * *OUTPUT_FILE*, void * *USER_DATA*)

Dump the internal structure of a queue to a file.

Dump the structure of the queue Q to OUTPUT_FILE. If WRITE_F != NULL, then uses WRITE_F to write Q's elements to OUTPUT_FILE. Additionnal USER_DATA argument could be passed to WRITE_F.

Note:
Complexity: O(|Q|)

Precondition:
Q must be a valid gdsl_queue_t & OUTPUT_FILE != NULL

Parameters:
Q The queue to write.
WRITE_F The write function.
OUTPUT_FILE The file where to write Q's elements.
USER_DATA User's datas passed to WRITE_F.

See also:
gdsl_queue_write()(p. 151)
gdsl_queue_write_xml()(p. 151)

3.12.2.3 void gdsl_queue_flush (gdsl_queue_t *Q*)

Flush a queue.

Deallocate all the elements of the queue Q by calling Q's FREE_F function passed to gdsl_queue_allocc(). Q is not deallocated itself and Q's name is not modified.

Note:
Complexity: O(|Q|)

Precondition:
Q must be a valid gdsl_queue_t

Parameters:
Q The queue to flush

See also:
gdsl_queue_alloc()(p. 143)
gdsl_queue_free()(p. 144)

3.12.2.4 void gdsl_queue_free (gdsl_queue_t *Q*)

Destroy a queue.

Deallocate all the elements of the queue Q by calling Q's FREE_F function passed to **gdsl_queue_alloc()**(p. 143). The name of Q is deallocated and Q is deallocated itself too.

Note:
Complexity: O(|Q|)

Precondition:
Q must be a valid gdsl_queue_t

Parameters:
Q The queue to destroy

See also:
gdsl_queue_alloc()(p. 143)
gdsl_queue_flush()(p. 144)

3.12.2.5 gdsl_element_t gdsl_queue_get_head (const gdsl_queue_t *Q*)

Get the head of a queue.

Note:
Complexity: O(1)

Precondition:
Q must be a valid gdsl_queue_t

Parameters:
Q The queue to get the head from

Returns:
the element contained at the header position of the queue Q if Q is not empty. The returned element is not removed from Q.
NULL if the queue Q is empty.

See also:
gdsl_queue_get_tail()(p. 146)

3.12.2.6 const char* gdsl_queue_get_name (const gdsl_queue_t *Q*)

Getsthe name of a queue.

Note:
Complexity: O(1)

Precondition:
Q must be a valid gdsl_queue_t

Postcondition:
The returned string MUST NOT be freed.

Parameters:
Q The queue to get the name from

Returns:
the name of the queue Q.

See also:
gdsl_queue_set_name()(p. 150)

3.12.2.7 ulong gdsl_queue_get_size (const gdsl_queue_t *Q*)

Get the size of a queue.

Note:
Complexity: O(1)

Precondition:
Q must be a valid gdsl_queue_t

Parameters:
Q The queue to get the size from

Returns:
the number of elements of Q (noted |Q|).

3.12.2.8 gdsl_element_t gdsl_queue_get_tail (const gdsl_queue_t *Q*)

Get the tail of a queue.

Note:
Complexity: O(1)

Precondition:
Q must be a valid gdsl_queue_t

Parameters:
Q The queue to get the tail from

Returns:
the element contained at the footer position of the queue Q if Q is not empty. The returned element is not removed from Q.
NULL if the queue Q is empty.

See also:
gdsl_queue_get_head()(p. 145)

3.12.2.9 gdsl_element_t gdsl_queue_insert (gdsl_queue_t *Q*, void * *VALUE*)

Insert an element in a queue (PUT).

Allocate a new element E by calling Q's ALLOC_F function on VALUE. ALLOC_F is the function pointer passed to **gdsl_queue_alloc()**(p. 143). The new element E is then inserted at the header position of the queue Q.

Note:
Complexity: O(1)

Precondition:
Q must be a valid gdsl_queue_t

Parameters:
Q The queue to insert in

VALUE The value used to make the new element to insert into Q

Returns:
the inserted element E in case of success.
NULL in case of insufficient memory.

See also:
gdsl_queue_remove()(p. 149)

3.12.2.10 bool gdsl_queue_is_empty (const gdsl_queue_t *Q*)

Check if a queue is empty.

Note:
Complexity: O(1)

Precondition:
Q must be a valid gdsl_queue_t

Parameters:
Q The queue to check

Returns:
TRUE if the queue Q is empty.
FALSE if the queue Q is not empty.

3.12.2.11 gdsl_element_t gdsl_queue_map_backward (const gdsl_queue_t *Q*, gdsl_map_func_t *MAP_F*, void * *USER_DATA*)

Parse a queue from tail to head.

Parse all elements of the queue Q from tail to head. The MAP_F function is called on each Q's element with USER_DATA argument. If MAP_F returns GDSL_MAP_STOP, then **gdsl_queue_map_backward()**(p. 148) stops and returns its last examinated element.

Note:
Complexity: O(|Q|)

Precondition:
Q must be a valid gdsl_queue_t & MAP_F != NULL

Parameters:
Q The queue to parse

MAP_F The map function to apply on each Q's element

USER_DATA User's datas passed to MAP_F Returns the first element for which MAP_F returns GDSL_MAP_STOP. Returns NULL when the parsing is done.

See also:
gdsl_queue_map_forward()(p. 148)

3.12.2.12 gdsl_element_t gdsl_queue_map_forward (const gdsl_queue_t *Q*, gdsl_map_func_t *MAP_F*, void * *USER_DATA*)

Parse a queue from head to tail.

Parse all elements of the queue Q from head to tail. The MAP_F function is called on each Q's element with USER_DATA argument. If MAP_F returns GDSL_MAP_STOP, then **gdsl_queue_map_forward()**(p. 148) stops and returns its last examinated element.

Note:
Complexity: O(|Q|)

Precondition:
Q must be a valid gdsl_queue_t & MAP_F != NULL

Parameters:
Q The queue to parse

MAP_F The map function to apply on each Q's element

USER_DATA User's datas passed to MAP_F

Returns:
the first element for which MAP_F returns GDSL_MAP_STOP.
NULL when the parsing is done.

See also:
gdsl_queue_map_backward()(p. 148)

3.12.2.13 gdsl_element_t gdsl_queue_remove (gdsl_queue_t *Q*)

Remove an element from a queue (GET).

Remove the element at the footer position of the queue Q.

Note:
Complexity: O(1)

Precondition:
Q must be a valid gdsl_queue_t

Parameters:
Q The queue to remove the tail from

Returns:
the removed element in case of success.
NULL in case of Q is empty.

See also:
gdsl_queue_insert()(p. 147)

3.12.2.14 gdsl_element_t gdsl_queue_search (const gdsl_queue_t *Q*, gdsl_compare_func_t *COMP_F*, void * *VALUE*)

Search for a particular element in a queue.

Search for the first element E equal to VALUE in the queue Q, by using COMP_F to compare all Q's element with.

Note:
Complexity: O(|Q| / 2)

Precondition:
Q must be a valid gdsl_queue_t & COMP_F != NULL

Parameters:
Q The queue to search the element in

COMP_F The comparison function used to compare Q's element with VALUE

VALUE The value to compare Q's elements with

Returns:
the first founded element E in case of success.
NULL in case the searched element E was not found.

See also:
gdsl_queue_search_by_position(p. 150)

3.12.2.15 gdsl_element_t gdsl_queue_search_by_position (const gdsl_queue_t *Q*, ulong *POS*)

Search for an element by its position in a queue.

Note:
Complexity: O(|Q| / 2)

Precondition:
Q must be a valid gdsl_queue_t & POS > 0 & POS <= |Q|

Parameters:
Q The queue to search the element in

POS The position where is the element to search

Returns:
the element at the POS-th position in the queue Q.
NULL if POS > |L| or POS <= 0.

See also:
gdsl_queue_search()(p. 149)

3.12.2.16 gdsl_queue_t gdsl_queue_set_name (gdsl_queue_t *Q*, const char * *NEW_NAME*)

Set the name of a queue.

Change the previous name of the queue Q to a copy of NEW_NAME.

Note:
Complexity: O(1)

Precondition:
Q must be a valid gdsl_queue_t

Parameters:
Q The queue to change the name

NEW_NAME The new name of Q

Returns:
the modified queue in case of success.
NULL in case of insufficient memory.

See also:
gdsl_queue_get_name()(p. 145)

3.12.2.17 void gdsl_queue_write (const gdsl_queue_t *Q*, gdsl_write_func_t *WRITE_F*, FILE * *OUTPUT_FILE*, void * *USER_DATA*)

Write all the elements of a queue to a file.

Write the elements of the queue Q to OUTPUT_FILE, using WRITE_F function. Additionnal USER_DATA argument could be passed to WRITE_F.

Note:
Complexity: O(|Q|)

Precondition:
Q must be a valid gdsl_queue_t & OUTPUT_FILE != NULL & WRITE_F != NULL

Parameters:
Q The queue to write.

WRITE_F The write function.

OUTPUT_FILE The file where to write Q's elements.

USER_DATA User's datas passed to WRITE_F.

See also:
gdsl_queue_write_xml()(p. 151)
gdsl_queue_dump()(p. 143)

3.12.2.18 void gdsl_queue_write_xml (const gdsl_queue_t *Q*, gdsl_write_func_t *WRITE_F*, FILE * *OUTPUT_FILE*, void * *USER_DATA*)

Write the content of a queue to a file into XML.

Write the elements of the queue Q to OUTPUT_FILE, into XML language. If WRITE_F != NULL, then uses WRITE_F to write Q's elements to OUTPUT_FILE. Additionnal USER_DATA argument could be passed to WRITE_F.

Note:
Complexity: O(|Q|)

Precondition:
Q must be a valid gdsl_queue_t & OUTPUT_FILE != NULL

Parameters:
Q The queue to write.
WRITE_F The write function.
OUTPUT_FILE The file where to write Q's elements.
USER_DATA User's datas passed to WRITE_F.

See also:
gdsl_queue_write()(p. 151)
gdsl_queue_dump()(p. 143)

3.13 Red-black tree manipulation module

Typedefs

- typedef gdsl_rbtree * **gdsl_rbtree_t**

Functions

- **gdsl_rbtree_t gdsl_rbtree_alloc** (const char *NAME, **gdsl_alloc_func_t** ALLOC_F, **gdsl_free_func_t** FREE_F, **gdsl_compare_func_t** COMP_F)

  Create a new red-black tree.

- void **gdsl_rbtree_free** (**gdsl_rbtree_t** T)

  Destroy a red-black tree.

- void **gdsl_rbtree_flush** (**gdsl_rbtree_t** T)

  Flush a red-black tree.

- char * **gdsl_rbtree_get_name** (const **gdsl_rbtree_t** T)

  Get the name of a red-black tree.

- **bool gdsl_rbtree_is_empty** (const **gdsl_rbtree_t** T)

  Check if a red-black tree is empty.

- **gdsl_element_t gdsl_rbtree_get_root** (const **gdsl_rbtree_t** T)

  Get the root of a red-black tree.

- **ulong gdsl_rbtree_get_size** (const **gdsl_rbtree_t** T)

  Get the size of a red-black tree.

- **ulong gdsl_rbtree_height** (const **gdsl_rbtree_t** T)

  Get the height of a red-black tree.

- **gdsl_rbtree_t gdsl_rbtree_set_name** (**gdsl_rbtree_t** T, const char *NEW_NAME)

  Set the name of a red-black tree.

- **gdsl_element_t gdsl_rbtree_insert** (**gdsl_rbtree_t** T, void *VALUE, int *RESULT)

  Insert an element into a red-black tree if it's not found or return it.

- **gdsl_element_t gdsl_rbtree_remove** (**gdsl_rbtree_t** T, void *VALUE)

  Remove an element from a red-black tree.

- **gdsl_rbtree_t gdsl_rbtree_delete** (**gdsl_rbtree_t** T, void *VALUE)

  Delete an element from a red-black tree.

- **gdsl_element_t gdsl_rbtree_search** (const **gdsl_rbtree_t** T, **gdsl_compare_func_t** COMP_F, void *VALUE)

  Search for a particular element into a red-black tree.

- **gdsl_element_t gdsl_rbtree_map_prefix** (const **gdsl_rbtree_t** T, **gdsl_map_func_t** MAP_F, void *USER_DATA)

  Parse a red-black tree in prefixed order.

- **gdsl_element_t gdsl_rbtree_map_infix** (const **gdsl_rbtree_t** T, **gdsl_map_func_t** MAP_F, void *USER_DATA)

  Parse a red-black tree in infixed order.

- **gdsl_element_t gdsl_rbtree_map_postfix** (const **gdsl_rbtree_t** T, **gdsl_map_func_t** MAP_F, void *USER_DATA)

  Parse a red-black tree in postfixed order.

- void **gdsl_rbtree_write** (const **gdsl_rbtree_t** T, **gdsl_write_func_t** WRITE_F, FILE *OUTPUT_FILE, void *USER_DATA)

  Write the element of each node of a red-black tree to a file.

- void **gdsl_rbtree_write_xml** (const **gdsl_rbtree_t** T, **gdsl_write_func_t** WRITE_F, FILE *OUTPUT_FILE, void *USER_DATA)

  Write the content of a red-black tree to a file into XML.

- void **gdsl_rbtree_dump** (const **gdsl_rbtree_t** T, **gdsl_write_func_t** WRITE_F, FILE *OUTPUT_FILE, void *USER_DATA)

  Dump the internal structure of a red-black tree to a file.

3.13.1 Typedef Documentation

3.13.1.1 typedef struct gdsl_rbtree* gdsl_rbtree_t

GDSL red-black tree type.

This type is voluntary opaque. Variables of this kind could'nt be directly used, but by the functions of this module.

Definition at line 51 of file gdsl_rbtree.h.

3.13.2 Function Documentation

3.13.2.1 gdsl_rbtree_t gdsl_rbtree_alloc (const char * *NAME*, gdsl_alloc_func_t *ALLOC_F*, gdsl_free_func_t *FREE_F*, gdsl_compare_func_t *COMP_F*)

Create a new red-black tree.

Allocate a new red-black tree data structure which name is set to a copy of NAME. The function pointers ALLOC_F, FREE_F and COMP_F could be used to respectively, alloc, free and compares elements in the tree. These pointers could be set to NULL to use the default ones:

- the default ALLOC_F simply returns its argument
- the default FREE_F does nothing
- the default COMP_F always returns 0

Note:
Complexity: O(1)

Precondition:
nothing

Parameters:
NAME The name of the new red-black tree to create
ALLOC_F Function to alloc element when inserting it in a r-b tree
FREE_F Function to free element when removing it from a r-b tree
COMP_F Function to compare elements into the r-b tree

Returns:
the newly allocated red-black tree in case of success.
NULL in case of failure.

See also:
gdsl_rbtree_free()(p. 157)
gdsl_rbtree_flush()(p. 157)

3.13.2.2 gdsl_rbtree_t gdsl_rbtree_delete (gdsl_rbtree_t *T*, void * *VALUE*)

Delete an element from a red-black tree.

Remove from the red-black tree the first founded element E equal to VALUE, by using T's COMP_F function passed to **gdsl_rbtree_alloc()**(p. 155). If E is found, it is removed from T and E is deallocated using T's FREE_F function passed to **gdsl_rbtree_alloc()**(p. 155), then T is returned.

Note:
Complexity: O(log(|T|))

Precondition:
T must be a valid gdsl_rbtree_t

Parameters:
T The red-black tree to remove an element from

VALUE The value used to find the element to remove

Returns:
the modified red-black tree after removal of E if E was found.
NULL if no element equal to VALUE was found.

See also:
gdsl_rbtree_insert()(p. 159)
gdsl_rbtree_remove()(p. 162)

3.13.2.3 void gdsl_rbtree_dump (const gdsl_rbtree_t *T*, gdsl_write_func_t *WRITE_F*, FILE * *OUTPUT_FILE*, void * *USER_DATA*)

Dump the internal structure of a red-black tree to a file.

Dump the structure of the red-black tree T to OUTPUT_FILE. If WRITE_F != NULL, then use WRITE_F to write T's nodes elements to OUTPUT_FILE. Additionnal USER_DATA argument could be passed to WRITE_F.

Note:
Complexity: O(|T|)

Precondition:
T must be a valid gdsl_rbtree_t & OUTPUT_FILE != NULL

Parameters:
T The red-black tree to write.

WRITE_F The write function.

OUTPUT_FILE The file where to write T's elements.

USER_DATA User's datas passed to WRITE_F.

See also:
gdsl_rbtree_write()(p. 164)
gdsl_rbtree_write_xml()(p. 164)

3.13.2.4 void gdsl_rbtree_flush (gdsl_rbtree_t *T*)

Flush a red-black tree.

Deallocate all the elements of the red-black tree T by calling T's FREE_F function passed to **gdsl_rbtree_alloc()**(p. 155). The red-black tree T is not deallocated itself and its name is not modified.

Note:
Complexity: O(|T|)

Precondition:
T must be a valid gdsl_rbtree_t

See also:
gdsl_rbtree_alloc()(p. 155)
gdsl_rbtree_free()(p. 157)

3.13.2.5 void gdsl_rbtree_free (gdsl_rbtree_t *T*)

Destroy a red-black tree.

Deallocate all the elements of the red-black tree T by calling T's FREE_F function passed to **gdsl_rbtree_alloc()**(p. 155). The name of T is deallocated and T is deallocated itself too.

Note:
Complexity: O(|T|)

Precondition:
T must be a valid gdsl_rbtree_t

Parameters:
T The red-black tree to deallocate

See also:
gdsl_rbtree_alloc()(p. 155)
gdsl_rbtree_flush()(p. 157)

3.13.2.6 char* gdsl_rbtree_get_name (const gdsl_rbtree_t *T*)

Get the name of a red-black tree.

Note:
Complexity: O(1)

Precondition:
T must be a valid gdsl_rbtree_t

Postcondition:
The returned string MUST NOT be freed.

Parameters:
T The red-black tree to get the name from

Returns:
the name of the red-black tree T.

See also:
gdsl_rbtree_set_name()(p. 163)

3.13.2.7 gdsl_element_t gdsl_rbtree_get_root (const gdsl_rbtree_t *T*)

Get the root of a red-black tree.

Note:
Complexity: O(1)

Precondition:
T must be a valid gdsl_rbtree_t

Parameters:
T The red-black tree to get the root element from

Returns:
the element at the root of the red-black tree T.

3.13.2.8 ulong gdsl_rbtree_get_size (const gdsl_rbtree_t *T*)

Get the size of a red-black tree.

Note:
Complexity: O(1)

Precondition:
T must be a valid gdsl_rbtree_t

Parameters:
T The red-black tree to get the size from

Returns:
the size of the red-black tree T (noted |T|).

See also:
gdsl_rbtree_get_height()

3.13.2.9 ulong gdsl_rbtree_height (const gdsl_rbtree_t *T*)

Get the height of a red-black tree.

Note:
Complexity: O(|T|)

Precondition:
T must be a valid gdsl_rbtree_t

Parameters:
T The red-black tree to compute the height from

Returns:
the height of the red-black tree T (noted h(T)).

See also:
gdsl_rbtree_get_size()(p. 158)

3.13.2.10 gdsl_element_t gdsl_rbtree_insert (gdsl_rbtree_t *T*, void * *VALUE*, int * *RESULT*)

Insert an element into a red-black tree if it's not found or return it.

Search for the first element E equal to VALUE into the red-black tree T, by using T's COMP_F function passed to gdsl_rbtree_alloc to find it. If E is found, then it's returned. If E isn't found, then a new element E is allocated using T's ALLOC_F function passed to gdsl_rbtree_alloc and is inserted and then returned.

Note:
Complexity: O(log(|T|))

Precondition:
T must be a valid gdsl_rbtree_t & RESULT != NULL

Parameters:
T The red-black tree to modify
VALUE The value used to make the new element to insert into T
RESULT The address where the result code will be stored.

Returns:
the element E and RESULT = GDSL_OK if E is inserted into T.
the element E and RESULT = GDSL_ERR_DUPLICATE_ENTRY if E is already present in T.
NULL and RESULT = GDSL_ERR_MEM_ALLOC in case of insufficient memory.

See also:
gdsl_rbtree_remove()(p. 162)
gdsl_rbtree_delete()(p. 155)

3.13.2.11 bool gdsl_rbtree_is_empty (const gdsl_rbtree_t *T*)

Check if a red-black tree is empty.

Note:
Complexity: O(1)

Precondition:
T must be a valid gdsl_rbtree_t

Parameters:
T The red-black tree to check

Returns:
TRUE if the red-black tree T is empty.
FALSE if the red-black tree T is not empty.

3.13.2.12 gdsl_element_t gdsl_rbtree_map_infix (const gdsl_rbtree_t *T*, gdsl_map_func_t *MAP_F*, void * *USER_DATA*)

Parse a red-black tree in infixed order.

Parse all nodes of the red-black tree T in infixed order. The MAP_F function is called on the element contained in each node with the USER_DATA argument. If MAP_F returns GDSL_MAP_STOP, then **gdsl_rbtree_map_infix()**(p. 160) stops and returns its last examinated element.

Note:
Complexity: O(|T|)

Precondition:
T must be a valid gdsl_rbtree_t & MAP_F != NULL

Parameters:
T The red-black tree to map.

MAP_F The map function.

USER_DATA User's datas passed to MAP_F

Returns:
the first element for which MAP_F returns GDSL_MAP_STOP.
NULL when the parsing is done.

See also:
gdsl_rbtree_map_prefix()(p. 161)
gdsl_rbtree_map_postfix()(p. 161)

3.13.2.13 gdsl_element_t gdsl_rbtree_map_postfix (const gdsl_rbtree_t *T*, gdsl_map_func_t *MAP_F*, void * *USER_DATA*)

Parse a red-black tree in postfixed order.

Parse all nodes of the red-black tree T in postfixed order. The MAP_F function is called on the element contained in each node with the USER_DATA argument. If MAP_F returns GDSL_MAP_STOP, then **gdsl_rbtree_map_postfix()**(p. 161) stops and returns its last examinated element.

Note:
Complexity: O(|T|)

Precondition:
T must be a valid gdsl_rbtree_t & MAP_F != NULL

Parameters:
T The red-black tree to map.
MAP_F The map function.
USER_DATA User's datas passed to MAP_F

Returns:
the first element for which MAP_F returns GDSL_MAP_STOP.
NULL when the parsing is done.

See also:
gdsl_rbtree_map_prefix()(p. 161)
gdsl_rbtree_map_infix()(p. 160)

3.13.2.14 gdsl_element_t gdsl_rbtree_map_prefix (const gdsl_rbtree_t *T*, gdsl_map_func_t *MAP_F*, void * *USER_DATA*)

Parse a red-black tree in prefixed order.

Parse all nodes of the red-black tree T in prefixed order. The MAP_F function is called on the element contained in each node with the USER_DATA argument. If MAP_F returns GDSL_MAP_STOP, then **gdsl_rbtree_map_prefix()**(p. 161) stops and returns its last examinated element.

Note:
Complexity: O(|T|)

Precondition:
T must be a valid gdsl_rbtree_t & MAP_F != NULL

Parameters:
T The red-black tree to map.

MAP_F The map function.

USER_DATA User's datas passed to MAP_F

Returns:
the first element for which MAP_F returns GDSL_MAP_STOP.
NULL when the parsing is done.

See also:
gdsl_rbtree_map_infix()(p. 160)
gdsl_rbtree_map_postfix()(p. 161)

3.13.2.15 gdsl_element_t gdsl_rbtree_remove (gdsl_rbtree_t *T*, void * *VALUE*)

Remove an element from a red-black tree.

Remove from the red-black tree T the first founded element E equal to VALUE, by using T's COMP_F function passed to **gdsl_rbtree_alloc()**(p. 155). If E is found, it is removed from T and then returned.

Note:
Complexity: O(log (|T|))

Precondition:
T must be a valid gdsl_rbtree_t

Parameters:
T The red-black tree to modify

VALUE The value used to find the element to remove

Returns:
the first founded element equal to VALUE in T in case is found.
NULL in case no element equal to VALUE is found in T.

See also:
gdsl_rbtree_insert()(p. 159)
gdsl_rbtree_delete()(p. 155)

3.13.2.16 gdsl_element_t gdsl_rbtree_search (const gdsl_rbtree_t *T*, gdsl_compare_func_t *COMP_F*, void * *VALUE*)

Search for a particular element into a red-black tree.

Search the first element E equal to VALUE in the red-black tree T, by using COMP_F function to find it. If COMP_F == NULL, then the COMP_F function passed to **gdsl_rbtree_alloc()**(p. 155) is used.

Note:
Complexity: O(log(|T|))

Precondition:
T must be a valid gdsl_rbtree_t

Parameters:
T The red-black tree to use.

COMP_F The comparison function to use to compare T's element with VALUE to find the element E (or NULL to use the default T's COMP_F)

VALUE The value that must be used by COMP_F to find the element E

Returns:
the first founded element E equal to VALUE.
NULL if VALUE is not found in T.

See also:
gdsl_rbtree_insert()(p. 159)
gdsl_rbtree_remove()(p. 162)
gdsl_rbtree_delete()(p. 155)

3.13.2.17 gdsl_rbtree_t gdsl_rbtree_set_name (gdsl_rbtree_t *T*, const char * *NEW_NAME*)

Set the name of a red-black tree.

Change the previous name of the red-black tree T to a copy of NEW_NAME.

Note:
Complexity: O(1)

Precondition:
T must be a valid gdsl_rbtree_t

Parameters:
T The red-black tree to change the name

NEW_NAME The new name of T

Returns:
the modified red-black tree in case of success.
NULL in case of insufficient memory.

See also:
gdsl_rbtree_get_name()(p. 157)

3.13.2.18 void gdsl_rbtree_write (const gdsl_rbtree_t *T*, gdsl_write_func_t *WRITE_F*, FILE * *OUTPUT_FILE*, void * *USER_DATA*)

Write the element of each node of a red-black tree to a file.

Write the nodes elements of the red-black tree T to OUTPUT_FILE, using WRITE_F function. Additionnal USER_DATA argument could be passed to WRITE_F.

Note:
Complexity: O(|T|)

Precondition:
T must be a valid gdsl_rbtree_t & WRITE_F != NULL & OUTPUT_FILE != NULL

Parameters:
T The red-black tree to write.

WRITE_F The write function.

OUTPUT_FILE The file where to write T's elements.

USER_DATA User's datas passed to WRITE_F.

See also:
gdsl_rbtree_write_xml()(p. 164)
gdsl_rbtree_dump()(p. 156)

3.13.2.19 void gdsl_rbtree_write_xml (const gdsl_rbtree_t *T*, gdsl_write_func_t *WRITE_F*, FILE * *OUTPUT_FILE*, void * *USER_DATA*)

Write the content of a red-black tree to a file into XML.

Write the nodes elements of the red-black tree T to OUTPUT_FILE, into XML language. If WRITE_F != NULL, then use WRITE_F to write T's nodes elements to OUTPUT_FILE. Additionnal USER_DATA argument could be passed to WRITE_F.

Note:
Complexity: O(|T|)

Precondition:
T must be a valid gdsl_rbtree_t & OUTPUT_FILE != NULL

Parameters:
T The red-black tree to write.

WRITE_F The write function.

OUTPUT_FILE The file where to write T's elements.

USER_DATA User's datas passed to WRITE_F.

See also:

gdsl_rbtree_write()(p. 164)
gdsl_rbtree_dump()(p. 156)

3.14 Sort module

Functions

- void **gdsl_sort** (**gdsl_element_t** *T, **ulong** N, **gdsl_compare_func_t** COMP_F)

  Sort an array in place.

3.14.1 Function Documentation

3.14.1.1 void gdsl_sort (gdsl_element_t * *T*, ulong *N*, gdsl_compare_func_t *COMP_F*)

Sort an array in place.

Sort the array T in place. The function COMP_F is used to compare T's elements and must be user-defined.

Note:
Complexity: O(N log(N))

Precondition:
N == |T| & T != NULL & COMP_F != NULL

Parameters:

T The array of elements to sort

N The number of elements into T

COMP_F The function pointer used to compare T's elements

3.15 Stack manipulation module

Typedefs

- typedef _gdsl_stack * **gdsl_stack_t**

  GDSL stack type.

Functions

- **gdsl_stack_t gdsl_stack_alloc** (const char *NAME, **gdsl_alloc_func_t** ALLOC_F, **gdsl_free_func_t** FREE_F)

  Create a new stack.

- void **gdsl_stack_free** (**gdsl_stack_t** S)

  Destroy a stack.

- void **gdsl_stack_flush** (**gdsl_stack_t** S)

  Flush a stack.

- const char * **gdsl_stack_get_name** (const **gdsl_stack_t** S)

  Getsthe name of a stack.

- **ulong gdsl_stack_get_size** (const **gdsl_stack_t** S)

  Get the size of a stack.

- ubyte **gdsl_stack_get_growing_factor** (const **gdsl_stack_t** S)

  Get the growing factor of a stack.

- **bool gdsl_stack_is_empty** (const **gdsl_stack_t** S)

  Check if a stack is empty.

- **gdsl_element_t gdsl_stack_get_top** (const **gdsl_stack_t** S)

  Get the top of a stack.

- **gdsl_element_t gdsl_stack_get_bottom** (const **gdsl_stack_t** S)

  Get the bottom of a stack.

- **gdsl_stack_t gdsl_stack_set_name** (**gdsl_stack_t** S, const char *NEW_NAME)

  Set the name of a stack.

- void **gdsl_stack_set_growing_factor** (**gdsl_stack_t** S, ubyte G)

  Set the growing factor of a stack.

- **gdsl_element_t gdsl_stack_insert** (**gdsl_stack_t** S, void *VALUE)

  Insert an element in a stack (PUSH).

- **gdsl_element_t gdsl_stack_remove** (**gdsl_stack_t** S)

  Remove an element from a stack (POP).

- **gdsl_element_t gdsl_stack_search** (const **gdsl_stack_t** S, **gdsl_compare_func_t** COMP_F, void *VALUE)

  Search for a particular element in a stack.

- **gdsl_element_t gdsl_stack_search_by_position** (const **gdsl_stack_t** S, **ulong** POS)

  Search for an element by its position in a stack.

- **gdsl_element_t gdsl_stack_map_forward** (const **gdsl_stack_t** S, **gdsl_map_func_t** MAP_F, void *USER_DATA)

  Parse a stack from bottom to top.

- **gdsl_element_t gdsl_stack_map_backward** (const **gdsl_stack_t** S, **gdsl_map_func_t** MAP_F, void *USER_DATA)

  Parse a stack from top to bottom.

- void **gdsl_stack_write** (const **gdsl_stack_t** S, **gdsl_write_func_t** WRITE_F, FILE *OUTPUT_FILE, void *USER_DATA)

  Write all the elements of a stack to a file.

- void **gdsl_stack_write_xml** (**gdsl_stack_t** S, **gdsl_write_func_t** WRITE_F, FILE *OUTPUT_FILE, void *USER_DATA)

  Write the content of a stack to a file into XML.

- void **gdsl_stack_dump** (**gdsl_stack_t** S, **gdsl_write_func_t** WRITE_F, FILE *OUTPUT_FILE, void *USER_DATA)

  Dump the internal structure of a stack to a file.

3.15.1 Typedef Documentation

3.15.1.1 typedef struct _gdsl_stack* gdsl_stack_t

GDSL stack type.

This type is voluntary opaque. Variables of this kind could'nt be directly used, but by the functions of this module.

Definition at line 52 of file gdsl_stack.h.

3.15.2 Function Documentation

3.15.2.1 gdsl_stack_t gdsl_stack_alloc (const char * *NAME*, gdsl_alloc_func_t *ALLOC_F*, gdsl_free_func_t *FREE_F*)

Create a new stack.

Allocate a new stack data structure which name is set to a copy of NAME. The functions pointers ALLOC_F and FREE_F could be used to respectively, alloc and free elements in the stack. These pointers could be set to NULL to use the default ones:

- the default ALLOC_F simply returns its argument
- the default FREE_F does nothing

Note:
Complexity: O(1)

Precondition:
nothing.

Parameters:
NAME The name of the new stack to create

ALLOC_F Function to alloc element when inserting it in a stack

FREE_F Function to free element when deleting it from a stack

Returns:
the newly allocated stack in case of success.
NULL in case of insufficient memory.

See also:
gdsl_stack_free()(p. 170)
gdsl_stack_flush()(p. 170)

3.15.2.2 void gdsl_stack_dump (gdsl_stack_t *S*, gdsl_write_func_t *WRITE_F*, FILE * *OUTPUT_FILE*, void * *USER_DATA*)

Dump the internal structure of a stack to a file.

Dump the structure of the stack S to OUTPUT_FILE. If WRITE_F != NULL, then uses WRITE_F to write S's elements to OUTPUT_FILE. Additionnal USER_DATA argument could be passed to WRITE_F.

Note:
Complexity: O(|S|)

Precondition:
S must be a valid gdsl_stack_t & OUTPUT_FILE != NULL

Parameters:
S The stack to write.

WRITE_F The write function.

OUTPUT_FILE The file where to write S's elements.

USER_DATA User's datas passed to WRITE_F.

See also:
gdsl_stack_write()(p. 178)
gdsl_stack_write_xml()(p. 178)

3.15.2.3 void gdsl_stack_flush (gdsl_stack_t *S*)

Flush a stack.

Deallocate all the elements of the stack S by calling S's FREE_F function passed to **gdsl_stack_alloc()**(p. 169). S is not deallocated itself and S's name is not modified.

Note:
Complexity: O(|S|)

Precondition:
S must be a valid gdsl_stack_t

Parameters:
S The stack to flush

See also:
gdsl_stack_alloc()(p. 169)
gdsl_stack_free()(p. 170)

3.15.2.4 void gdsl_stack_free (gdsl_stack_t *S*)

Destroy a stack.

Deallocate all the elements of the stack S by calling S's FREE_F function passed to **gdsl_stack_alloc()**(p. 169). The name of S is deallocated and S is deallocated itself too.

Note:
Complexity: O(|S|)

Precondition:
S must be a valid gdsl_stack_t

Parameters:
S The stack to destroy

See also:
gdsl_stack_alloc()(p. 169)
gdsl_stack_flush()(p. 170)

3.15.2.5 gdsl_element_t gdsl_stack_get_bottom (const gdsl_stack_t *S*)

Get the bottom of a stack.

Note:
Complexity: O(1)

Precondition:
S must be a valid gdsl_stack_t

Parameters:
S The stack to get the bottom from

Returns:
the element contained at the bottom position of the stack S if S is not empty. The returned element is not removed from S.
NULL if the stack S is empty.

See also:
gdsl_stack_get_top()(p. 173)

3.15.2.6 ubyte gdsl_stack_get_growing_factor (const gdsl_stack_t *S*)

Get the growing factor of a stack.

Get the growing factor of the stack S. This value is the amount of cells to reserve for next insertions. For example, if you set this value to 10, each time the number of elements of S reaches 10, then 10 new cells will be reserved for next 10 insertions. It is a way to save time for insertions. This value is 1 by default and can be modified with **gdsl_stack_set_growing_factor()**(p. 177).

Note:
Complexity: O(1)

Precondition:
S must be a valid gdsl_stack_t

Parameters:
S The stack to get the growing factor from

Returns:
the growing factor of the stack S.

See also:
gdsl_stack_insert()(p. 173)
gdsl_stack_set_growing_factor()(p. 177)

3.15.2.7 const char* gdsl_stack_get_name (const gdsl_stack_t *S*)

Getsthe name of a stack.

Note:
Complexity: O(1)

Precondition:
Q must be a valid gdsl_stack_t

Postcondition:
The returned string MUST NOT be freed.

Parameters:
S The stack to get the name from

Returns:
the name of the stack S.

See also:
gdsl_stack_set_name()(p. 177)

3.15.2.8 ulong gdsl_stack_get_size (const gdsl_stack_t *S*)

Get the size of a stack.

Note:
Complexity: O(1)

Precondition:
S must be a valid gdsl_stack_t

Parameters:
S The stack to get the size from

Returns:
the number of elements of the stack S (noted |S|).

3.15.2.9 gdsl_element_t gdsl_stack_get_top (const gdsl_stack_t *S*)

Get the top of a stack.

Note:
Complexity: O(1)

Precondition:
S must be a valid gdsl_stack_t

Parameters:
S The stack to get the top from

Returns:
the element contained at the top position of the stack S if S is not empty. The returned element is not removed from S.
NULL if the stack S is empty.

See also:
gdsl_stack_get_bottom()(p. 171)

3.15.2.10 gdsl_element_t gdsl_stack_insert (gdsl_stack_t *S*, void * *VALUE*)

Insert an element in a stack (PUSH).

Allocate a new element E by calling S's ALLOC_F function on VALUE. ALLOC_F is the function pointer passed to **gdsl_stack_alloc()**(p. 169). The new element E is the inserted at the top position of the stack S. If the number of elements in S reaches S's growing factor (G), then G new cells are reserved for future insertions into S to save time.

Note:
Complexity: O(1)

Precondition:
S must be a valid gdsl_stack_t

Parameters:
S The stack to insert in
VALUE The value used to make the new element to insert into S

Returns:
the inserted element E in case of success.
NULL in case of insufficient memory.

See also:
gdsl_stack_set_growing_factor()(p. 177)
gdsl_stack_get_growing_factor()(p. 171)
gdsl_stack_remove()(p. 175)

3.15.2.11 bool gdsl_stack_is_empty (const gdsl_stack_t *S*)

Check if a stack is empty.

Note:
Complexity: O(1)

Precondition:
S must be a valid gdsl_stack_t

Parameters:
S The stack to check

Returns:
TRUE if the stack S is empty.
FALSE if the stack S is not empty.

3.15.2.12 gdsl_element_t gdsl_stack_map_backward (const gdsl_stack_t *S*, gdsl_map_func_t *MAP_F*, void * *USER_DATA*)

Parse a stack from top to bottom.

Parse all elements of the stack S from top to bottom. The MAP_F function is called on each S's element with USER_DATA argument. If MAP_F returns GDSL_MAP_STOP, then **gdsl_stack_map_backward()**(p. 174) stops and returns its last examinated element.

Note:
Complexity: O(|S|)

Precondition:
S must be a valid gdsl_stack_t & MAP_F != NULL

Parameters:
S The stack to parse

MAP_F The map function to apply on each S's element

USER_DATA User's datas passed to MAP_F

Returns:
the first element for which MAP_F returns GDSL_MAP_STOP.
NULL when the parsing is done.

See also:
gdsl_stack_map_forward()(p. 175)

3.15.2.13 gdsl_element_t gdsl_stack_map_forward (const gdsl_stack_t *S*, gdsl_map_func_t *MAP_F*, void * *USER_DATA*)

Parse a stack from bottom to top.

Parse all elements of the stack S from bottom to top. The MAP_F function is called on each S's element with USER_DATA argument. If MAP_F returns GDSL_MAP_STOP, then **gdsl_stack_map_forward()**(p. 175) stops and returns its last examinated element.

Note:
Complexity: O(|S|)

Precondition:
S must be a valid gdsl_stack_t & MAP_F != NULL

Parameters:
S The stack to parse

MAP_F The map function to apply on each S's element

USER_DATA User's datas passed to MAP_F Returns the first element for which MAP_F returns GDSL_MAP_STOP. Returns NULL when the parsing is done.

See also:
gdsl_stack_map_backward()(p. 174)

3.15.2.14 gdsl_element_t gdsl_stack_remove (gdsl_stack_t *S*)

Remove an element from a stack (POP).

Remove the element at the top position of the stack S.

Note:
Complexity: O(1)

Precondition:
S must be a valid gdsl_stack_t

Parameters:
S The stack to remove the top from

Returns:
the removed element in case of success.
NULL in case of S is empty.

See also:
gdsl_stack_insert()(p. 173)

3.15.2.15 gdsl_element_t gdsl_stack_search (const gdsl_stack_t *S*, gdsl_compare_func_t *COMP_F*, void * *VALUE*)

Search for a particular element in a stack.

Search for the first element E equal to VALUE in the stack S, by using COMP_F to compare all S's element with.

Note:
Complexity: O(|S|)

Precondition:
S must be a valid gdsl_stack_t & COMP_F != NULL

Parameters:
S The stack to search the element in

COMP_F The comparison function used to compare S's element with VALUE

VALUE The value to compare S's elements with

Returns:
the first founded element E in case of success.
NULL if no element is found.

See also:
gdsl_stack_search_by_position()(p. 176)

3.15.2.16 gdsl_element_t gdsl_stack_search_by_position (const gdsl_stack_t *S*, ulong *POS*)

Search for an element by its position in a stack.

Note:
Complexity: O(1)

Precondition:
S must be a valid gdsl_stack_t & POS > 0 & POS <= |S|

Parameters:
S The stack to search the element in

POS The position where is the element to search

Returns:
the element at the POS-th position in the stack S.
NULL if POS > |L| or POS <= 0.

See also:
gdsl_stack_search()(p. 176)

3.15.2.17 void gdsl_stack_set_growing_factor (gdsl_stack_t *S*, ubyte *G*)

Set the growing factor of a stack.

Set the growing factor of the stack S. This value is the amount of cells to reserve for next insertions. For example, if you set this value to 10, each time the number of elements of S reaches 10, then 10 new cells will be reserved for next 10 insertions. It is a way to save time for insertions. To know the actual value of the growing factor, use **gdsl_stack_get_growing_factor()**(p. 171)

Note:
Complexity: O(1)

Precondition:
S must be a valid gdsl_stack_t

Parameters:
S The stack to get the growing factor from
G The new growing factor of S.

Returns:
the growing factor of the stack S.

See also:
gdsl_stack_insert()(p. 173)
gdsl_stack_get_growing_factor()(p. 171)

3.15.2.18 gdsl_stack_t gdsl_stack_set_name (gdsl_stack_t *S*, const char * *NEW_NAME*)

Set the name of a stack.

Change the previous name of the stack S to a copy of NEW_NAME.

Note:
Complexity: O(1)

Precondition:
S must be a valid gdsl_stack_t

Parameters:
S The stack to change the name
NEW_NAME The new name of S

Returns:
the modified stack in case of success.
NULL in case of insufficient memory.

See also:
gdsl_stack_get_name()(p. 172)

3.15.2.19 void gdsl_stack_write (const gdsl_stack_t *S*, gdsl_write_func_t *WRITE_F*, FILE * *OUTPUT_FILE*, void * *USER_DATA*)

Write all the elements of a stack to a file.

Write the elements of the stack S to OUTPUT_FILE, using WRITE_F function. Additionnal USER_DATA argument could be passed to WRITE_F.

Note:
Complexity: O(|S|)

Precondition:
S must be a valid gdsl_stack_t & OUTPUT_FILE != NULL & WRITE_F != NULL

Parameters:
S The stack to write.

WRITE_F The write function.

OUTPUT_FILE The file where to write S's elements.

USER_DATA User's datas passed to WRITE_F.

See also:
gdsl_stack_write_xml()(p. 178)
gdsl_stack_dump()(p. 169)

3.15.2.20 void gdsl_stack_write_xml (gdsl_stack_t *S*, gdsl_write_func_t *WRITE_F*, FILE * *OUTPUT_FILE*, void * *USER_DATA*)

Write the content of a stack to a file into XML.

Write the elements of the stack S to OUTPUT_FILE, into XML language. If WRITE_F != NULL, then uses WRITE_F to write S's elements to OUTPUT_FILE. Additionnal USER_DATA argument could be passed to WRITE_F.

Note:
Complexity: O(|S|)

Precondition:
S must be a valid gdsl_stack_t & OUTPUT_FILE != NULL

Parameters:
S The stack to write.

WRITE_F The write function.

OUTPUT_FILE The file where to write S's elements.

USER_DATA User's datas passed to WRITE_F.

See also:
gdsl_stack_write()(p. 178)
gdsl_stack_dump()(p. 169)

3.16 GDSL types

Typedefs

- typedef void * **gdsl_element_t**

  GDSL element type.

- typedef **gdsl_element_t**(* **gdsl_alloc_func_t**)(void *USER_DATA)

  GDSL Alloc element function type.

- typedef void(* **gdsl_free_func_t**)(**gdsl_element_t** E)

  GDSL Free element function type.

- typedef **gdsl_element_t**(* **gdsl_copy_func_t**)(const **gdsl_element_t** E)

  GDSL Copy element function type.

- typedef int(* **gdsl_map_func_t**)(const **gdsl_element_t** E, void *USER_DATA)

  GDSL Map element function type.

- typedef int(* **gdsl_compare_func_t**)(const **gdsl_element_t** E, void *VALUE)

  GDSL Comparison element function type.

- typedef void(* **gdsl_write_func_t**)(const **gdsl_element_t** E, FILE *OUTPUT_FILE, void *USER_DATA)

  GDSL Write element function type.

- typedef unsigned long int **ulong**

Enumerations

- enum **gdsl_constant_t** {

  GDSL_ERR_MEM_ALLOC = -1, **GDSL_MAP_STOP** = 0, **GDSL_MAP_CONT** = 1, **GDSL_INSERTED**,

  GDSL_FOUND }

  GDSL Constants.

- enum **bool** { **FALSE** = 0, **TRUE** = 1 }

3.16.1 Typedef Documentation

3.16.1.1 typedef gdsl_element_t(* gdsl_alloc_func_t)(void* USER_DATA)

GDSL Alloc element function type.

This function type is for allocating a new gdsl_element_t variable. The USER_DATA argument should be used to fill-in the new element.

Parameters:

USER_DATA user data used to create the new element

Returns:

the newly allocated element in case of success.
NULL in case of failure.

See also:

gdsl_free_func_t(p. 182)

Definition at line 86 of file gdsl_types.h.

3.16.1.2 typedef int(* gdsl_compare_func_t)(const gdsl_element_t E, void* VALUE)

GDSL Comparison element function type.

This function type is used to compare a gdsl_element_t variable with a user value. The E argument is always the one in the GDSL data structure, VALUE is always the one the user wants to compare E with.

Parameters:

E The gdsl_element_t variable contained into the data structure to compare from

VALUE The user data to compare E with

Returns:

< 0 if E is assumed to be less than VALUE.
0 if E is assumed to be equal to VALUE.
> 0 if E is assumed to be greather than VALUE.

Definition at line 153 of file gdsl_types.h.

3.16.1.3 typedef gdsl_element_t(* gdsl_copy_func_t)(const gdsl_element_t E)

GDSL Copy element function type.

This function type is for copying gdsl_element_t variables.

Parameters:

E The gdsl_element_t variable to copy

Returns:

the copied element in case of success.
NULL in case of failure.

Definition at line 117 of file gdsl_types.h.

3.16.1.4 typedef void* gdsl_element_t

GDSL element type.

All GDSL internal data structures contains a field of this type. This field is for GDSL users to store their data into GDSL data structures.

Definition at line 72 of file gdsl_types.h.

3.16.1.5 typedef void(* gdsl_free_func_t)(gdsl_element_t E)

GDSL Free element function type.

This function type is for freeing a gdsl_element_t variable. The element must have been previously allocated by a function of gdsl_alloc_func_t type. A free function according to gdsl_free_func_t must free the ressources allocated by the corresponding call to the function of type gdsl_alloc_func_t. The GDSL functions doesn't check if E != NULL before calling this function.

Parameters:

E The element to deallocate

See also:

gdsl_alloc_func_t(p. 181)

Definition at line 104 of file gdsl_types.h.

3.16.1.6 typedef int(* gdsl_map_func_t)(const gdsl_element_t E, void* USER_DATA)

GDSL Map element function type.

This function type is for mapping a gdsl_element_t variable from a GDSL data structure. The optional USER_DATA could be used to do special thing if needed.

Parameters:

E The actually mapped gdsl_element_t variable

USER_DATA User's datas

Returns:
GDSL_MAP_STOP if the parsing must be stopped.
GDSL_MAP_CONT if the parsing must continue.

Definition at line 133 of file gdsl_types.h.

3.16.1.7 typedef void(* gdsl_write_func_t)(const gdsl_element_t E, FILE* OUTPUT_FILE, void* USER_DATA)

GDSL Write element function type.

This function type is for writing a gdsl_element_t E to OUTPUT_FILE. Additional USER_DATA could be passed to it.

Parameters:
E The gdsl_element_t variable to write
OUTPUT_FILE The FILE where to write E
USER_DATA User's datas

Definition at line 168 of file gdsl_types.h.

3.16.1.8 typedef unsigned long int ulong

Definition at line 187 of file gdsl_types.h.

3.16.2 Enumeration Type Documentation

3.16.2.1 enum bool

GDSL boolean type. Defines _NO_LIBGDSL_TYPES_ at compilation time if you don't want them.

Enumeration values:
FALSE FALSE boolean value
TRUE TRUE boolean value

Definition at line 210 of file gdsl_types.h.

3.16.2.2 enum gdsl_constant_t

GDSL Constants.

Enumeration values:
GDSL_ERR_MEM_ALLOC Memory allocation error
GDSL_MAP_STOP For stopping a parsing function
GDSL_MAP_CONT For continuing a parsing function

GDSL_INSERTED To indicate an inserted value

GDSL_FOUND To indicate a founded value

Definition at line 47 of file gdsl_types.h.

Chapter 4

GDSL File Documentation

4.1 _gdsl_bintree.h File Reference

Typedefs

- typedef _gdsl_bintree * **_gdsl_bintree_t**
  GDSL low-level binary tree type.
- typedef int(* **gdsl_bintree_map_func_t**)(**_gdsl_bintree_t** TREE, void *USER_DATA)
  GDSL low-level binary tree map function type.

Functions

- **_gdsl_bintree_t _gdsl_bintree_alloc** (const **gdsl_element_t** E, const **_gdsl_bintree_t** LEFT, const **_gdsl_bintree_t** RIGHT)
  Create a new low-level binary tree.
- void **_gdsl_bintree_free** (**_gdsl_bintree_t** T, const **gdsl_free_-func_t** FREE_F)
  Destroy a low-level binary tree.
- **_gdsl_bintree_t _gdsl_bintree_copy** (const **_gdsl_bintree_t** T, const **gdsl_copy_func_t** COPY_F)
  Copy a low-level binary tree.
- **bool _gdsl_bintree_is_empty** (const **_gdsl_bintree_t** T)
  Check if a low-level binary tree is empty.
- **bool _gdsl_bintree_is_leaf** (const **_gdsl_bintree_t** T)
  Check if a low-level binary tree is reduced to a leaf.

- **bool _gdsl_bintree_is_root** (const **_gdsl_bintree_t** T)

  Check if a low-level binary tree is a root.

- **gdsl_element_t _gdsl_bintree_get_content** (const **_gdsl_bintree_t** T)

  Get the root content of a low-level binary tree.

- **_gdsl_bintree_t _gdsl_bintree_get_parent** (const **_gdsl_bintree_t** T)

  Get the parent tree of a low-level binary tree.

- **_gdsl_bintree_t _gdsl_bintree_get_left** (const **_gdsl_bintree_t** T)

  Get the left sub-tree of a low-level binary tree.

- **_gdsl_bintree_t _gdsl_bintree_get_right** (const **_gdsl_bintree_t** T)

  Get the right sub-tree of a low-level binary tree.

- **_gdsl_bintree_t** * **_gdsl_bintree_get_left_ref** (const **_gdsl_bintree_t** T)

  Get the left sub-tree reference of a low-level binary tree.

- **_gdsl_bintree_t** * **_gdsl_bintree_get_right_ref** (const **_gdsl_bintree_t** T)

  Get the right sub-tree reference of a low-level binary tree.

- **ulong _gdsl_bintree_get_height** (const **_gdsl_bintree_t** T)

  Get the height of a low-level binary tree.

- **ulong _gdsl_bintree_get_size** (const **_gdsl_bintree_t** T)

  Get the size of a low-level binary tree.

- void **_gdsl_bintree_set_content** (**_gdsl_bintree_t** T, const **gdsl_element_t** E)

  Set the root element of a low-level binary tree.

- void **_gdsl_bintree_set_parent** (**_gdsl_bintree_t** T, const **_gdsl_bintree_t** P)

  Set the parent tree of a low-level binary tree.

- void **_gdsl_bintree_set_left** (**_gdsl_bintree_t** T, const **_gdsl_bintree_t** L)

  Set left sub-tree of a low-level binary tree.

- void **_gdsl_bintree_set_right** (**_gdsl_bintree_t** T, const **_gdsl_bintree_t** R)

  Set right sub-tree of a low-level binary tree.

- **_gdsl_bintree_t _gdsl_bintree_rotate_left** (**_gdsl_bintree_t** *T)

  Left rotate a low-level binary tree.

- **_gdsl_bintree_t _gdsl_bintree_rotate_right** (**_gdsl_bintree_t** *T)

  Right rotate a low-level binary tree.

- **_gdsl_bintree_t _gdsl_bintree_rotate_left_right** (**_gdsl_bintree_t** *T)

  Left-right rotate a low-level binary tree.

- **_gdsl_bintree_t _gdsl_bintree_rotate_right_left** (**_gdsl_bintree_t** *T)

  Right-left rotate a low-level binary tree.

- **_gdsl_bintree_t _gdsl_bintree_map_prefix** (const **_gdsl_bintree_t** T, const **gdsl_bintree_map_func_t** MAP_F, void *USER_DATA)

  Parse a low-level binary tree in prefixed order.

- **_gdsl_bintree_t _gdsl_bintree_map_infix** (const **_gdsl_bintree_t** T, const **gdsl_bintree_map_func_t** MAP_F, void *USER_DATA)

  Parse a low-level binary tree in infixed order.

- **_gdsl_bintree_t _gdsl_bintree_map_postfix** (const **_gdsl_bintree_t** T, const **gdsl_bintree_map_func_t** MAP_F, void *USER_DATA)

  Parse a low-level binary tree in postfixed order.

- void **_gdsl_bintree_write** (const **_gdsl_bintree_t** T, const **gdsl_write_func_t** WRITE_F, FILE *OUTPUT_FILE, void *USER_DATA)

  Write the content of all nodes of a low-level binary tree to a file.

- void **_gdsl_bintree_write_xml** (const **_gdsl_bintree_t** T, const **gdsl_write_func_t** WRITE_F, FILE *OUTPUT_FILE, void *USER_DATA)

  Write the content of a low-level binary tree to a file into XML.

- void **_gdsl_bintree_dump** (const **_gdsl_bintree_t** T, const **gdsl_write_func_t** WRITE_F, FILE *OUTPUT_FILE, void *USER_DATA)

Dump the internal structure of a low-level binary tree to a file.

4.2 _gdsl_bstree.h File Reference

Typedefs

- typedef **_gdsl_bintree_t _gdsl_bstree_t**

  GDSL low-level binary search tree type.

- typedef int(* **gdsl_bstree_map_func_t**)(**_gdsl_bstree_t** TREE, void *USER_DATA)

  GDSL low-level binary search tree map function type.

Functions

- **_gdsl_bstree_t _gdsl_bstree_alloc** (const **gdsl_element_t** E)

  Create a new low-level binary search tree.

- void **_gdsl_bstree_free** (**_gdsl_bstree_t** T, const **gdsl_free_-func_t** FREE_F)

  Destroy a low-level binary search tree.

- **_gdsl_bstree_t _gdsl_bstree_copy** (const **_gdsl_bstree_t** T, const **gdsl_copy_func_t** COPY_F)

  Copy a low-level binary search tree.

- **bool _gdsl_bstree_is_empty** (const **_gdsl_bstree_t** T)

  Check if a low-level binary search tree is empty.

- **bool _gdsl_bstree_is_leaf** (const **_gdsl_bstree_t** T)

  Check if a low-level binary search tree is reduced to a leaf.

- **gdsl_element_t _gdsl_bstree_get_content** (const **_gdsl_-bstree_t** T)

  Get the root content of a low-level binary search tree.

- **bool _gdsl_bstree_is_root** (const **_gdsl_bstree_t** T)

  Check if a low-level binary search tree is a root.

- **_gdsl_bstree_t _gdsl_bstree_get_parent** (const **_gdsl_-bstree_t** T)

  Get the parent tree of a low-level binary search tree.

- **_gdsl_bstree_t _gdsl_bstree_get_left** (const **_gdsl_bstree_t** T)

  Get the left sub-tree of a low-level binary search tree.

- **_gdsl_bstree_t _gdsl_bstree_get_right** (const **_gdsl_bstree_t** T)

  Get the right sub-tree of a low-level binary search tree.

- **ulong _gdsl_bstree_get_size** (const **_gdsl_bstree_t** T)

  Get the size of a low-level binary search tree.

- **ulong _gdsl_bstree_get_height** (const **_gdsl_bstree_t** T)

  Get the height of a low-level binary search tree.

- **_gdsl_bstree_t _gdsl_bstree_insert** (**_gdsl_bstree_t** *T, const **gdsl_compare_func_t** COMP_F, const **gdsl_element_t** VALUE, int *RESULT)

  Insert an element into a low-level binary search tree if it's not found or return it.

- **gdsl_element_t _gdsl_bstree_remove** (**_gdsl_bstree_t** *T, const **gdsl_compare_func_t** COMP_F, const **gdsl_element_t** VALUE)

  Remove an element from a low-level binary search tree.

- **_gdsl_bstree_t _gdsl_bstree_search** (const **_gdsl_bstree_t** T, const **gdsl_compare_func_t** COMP_F, const **gdsl_element_t** VALUE)

  Search for a particular element into a low-level binary search tree.

- **_gdsl_bstree_t _gdsl_bstree_search_next** (const **_gdsl_bstree_t** T, const **gdsl_compare_func_t** COMP_F, const **gdsl_element_t** VALUE)

  Search for the next element of a particular element into a low-level binary search tree, according to the binary search tree order.

- **_gdsl_bstree_t _gdsl_bstree_map_prefix** (const **_gdsl_bstree_t** T, const **gdsl_bstree_map_func_t** MAP_F, void *USER_DATA)

  Parse a low-level binary search tree in prefixed order.

- **_gdsl_bstree_t _gdsl_bstree_map_infix** (const **_gdsl_bstree_t** T, const **gdsl_bstree_map_func_t** MAP_F, void *USER_DATA)

  Parse a low-level binary search tree in infixed order.

- **_gdsl_bstree_t _gdsl_bstree_map_postfix** (const **_gdsl_bstree_t** T, const **gdsl_bstree_map_func_t** MAP_F, void *USER_DATA)

  Parse a low-level binary search tree in postfixed order.

- void **_gdsl_bstree_write** (const **_gdsl_bstree_t** T, const **gdsl_write_func_t** WRITE_F, FILE *OUTPUT_FILE, void *USER_DATA)

  Write the content of all nodes of a low-level binary search tree to a file.

- void **_gdsl_bstree_write_xml** (const **_gdsl_bstree_t** T, const **gdsl_write_func_t** WRITE_F, FILE *OUTPUT_FILE, void *USER_DATA)

  Write the content of a low-level binary search tree to a file into XML.

- void **_gdsl_bstree_dump** (const **_gdsl_bstree_t** T, const **gdsl_write_func_t** WRITE_F, FILE *OUTPUT_FILE, void *USER_DATA)

  Dump the internal structure of a low-level binary search tree to a file.

4.3 _gdsl_list.h File Reference

Typedefs

- typedef **_gdsl_node_t _gdsl_list_t**
  GDSL low-level doubly-linked list type.
- typedef int(* **_gdsl_list_map_func_t**)(**_gdsl_node_t** NODE, void *USER_DATA)
  GDSL low-level doubly-linked list map function type.

Functions

- **_gdsl_list_t _gdsl_list_alloc** (const **gdsl_element_t** E)
  Create a new low-level list.
- void **_gdsl_list_free** (**_gdsl_list_t** L, const **gdsl_free_func_t** FREE_F)
  Destroy a low-level list.
- **bool _gdsl_list_is_empty** (const **_gdsl_list_t** L)
  Check if a low-level list is empty.
- **ulong _gdsl_list_get_size** (const **_gdsl_list_t** L)
  Get the size of a low-level list.
- void **_gdsl_list_link** (**_gdsl_list_t** L1, **_gdsl_list_t** L2)
  Link two low-level lists together.
- void **_gdsl_list_insert_after** (**_gdsl_list_t** L, **_gdsl_list_t** PREV)
  Insert a low-level list after another one.
- void **_gdsl_list_insert_before** (**_gdsl_list_t** L, **_gdsl_list_t** SUCC)
  Insert a low-level list before another one.
- void **_gdsl_list_remove** (**_gdsl_node_t** NODE)
  Remove a node from a low-level list.
- **_gdsl_list_t _gdsl_list_search** (**_gdsl_list_t** L, const **gdsl_compare_func_t** COMP_F, void *VALUE)
  Search for a particular node in a low-level list.
- **_gdsl_list_t _gdsl_list_map_forward** (const **_gdsl_list_t** L, const **_gdsl_list_map_func_t** MAP_F, void *USER_DATA)

Parse a low-level list in forward order.

- **_gdsl_list_t _gdsl_list_map_backward** (const **_gdsl_list_t** L, const **_gdsl_list_map_func_t** MAP_F, void *USER_DATA)

  Parse a low-level list in backward order.

- void **_gdsl_list_write** (const **_gdsl_list_t** L, const **gdsl_write_func_t** WRITE_F, FILE *OUTPUT_FILE, void *USER_DATA)

  Write the contents of all nodes of a low-level list to a file.

- void **_gdsl_list_write_xml** (const **_gdsl_list_t** L, const **gdsl_write_func_t** WRITE_F, FILE *OUTPUT_FILE, void *USER_DATA)

  Write the contents of all nodes of a low-level list to a file into XML.

- void **_gdsl_list_dump** (const **_gdsl_list_t** L, const **gdsl_write_func_t** WRITE_F, FILE *OUTPUT_FILE, void *USER_DATA)

  Dump the internal structure of a low-level list to a file.

4.4 _gdsl_node.h File Reference

Typedefs

- typedef _gdsl_node * **_gdsl_node_t**

  GDSL low-level doubly linked node type.

Functions

- **_gdsl_node_t _gdsl_node_alloc** (void)

  Create a new low-level node.

- **gdsl_element_t _gdsl_node_free** (**_gdsl_node_t** NODE)

  Destroy a low-level node.

- **_gdsl_node_t _gdsl_node_get_succ** (const **_gdsl_node_t** NODE)

  Get the successor of a low-level node.

- **_gdsl_node_t _gdsl_node_get_pred** (const **_gdsl_node_t** NODE)

  Get the predecessor of a low-level node.

- **gdsl_element_t _gdsl_node_get_content** (const **_gdsl_node_t** NODE)

  Get the content of a low-level node.

- void **_gdsl_node_set_succ** (**_gdsl_node_t** NODE, const **_gdsl_node_t** SUCC)

  Set the successor of a low-level node.

- void **_gdsl_node_set_pred** (**_gdsl_node_t** NODE, const **_gdsl_node_t** PRED)

  Set the predecessor of a low-level node.

- void **_gdsl_node_set_content** (**_gdsl_node_t** NODE, const **gdsl_element_t** CONTENT)

  Set the content of a low-level node.

- void **_gdsl_node_link** (**_gdsl_node_t** NODE1, **_gdsl_node_t** NODE2)

  Link two low-level nodes together.

- void **_gdsl_node_unlink** (**_gdsl_node_t** NODE1, **_gdsl_node_t** NODE2)

Unlink two low-level nodes.

- void **_gdsl_node_write** (const **_gdsl_node_t** NODE, const **gdsl_write_func_t** WRITE_F, FILE *OUTPUT_FILE, void *USER_DATA)

  Write the content of a low-level node to a file.

- void **_gdsl_node_write_xml** (const **_gdsl_node_t** NODE, const **gdsl_write_func_t** WRITE_F, FILE *OUTPUT_FILE, void *USER_DATA)

  Write the content of a low-level node to a file into XML.

- void **_gdsl_node_dump** (const **_gdsl_node_t** NODE, const **gdsl_write_func_t** WRITE_F, FILE *OUTPUT_FILE, void *USER_DATA)

  Dump the internal structure of a low-level node to a file.

4.5 gdsl.h File Reference

Functions

- const char * **gdsl_get_version** (void)
  Get GDSL version number as a string.

4.6 gdsl_2darray.h File Reference

Typedefs

- typedef gdsl_2darray * **gdsl_2darray_t**

  GDSL 2D-array type.

- typedef void(* **gdsl_2darray_write_func_t**)(**gdsl_element_t** E, const FILE *OUTPUT_FILE, **gdsl_2darray_position_t** POSITION, void *USER_DATA)

  GDSL 2D-array write function type.

Enumerations

- enum **gdsl_2darray_position_t** { **GDSL_2DARRAY_POSITION_FIRST_ROW** = 1, **GDSL_2DARRAY_POSITION_LAST_ROW** = 2, **GDSL_2DARRAY_POSITION_FIRST_COL** = 4, **GDSL_2DARRAY_POSITION_LAST_COL** = 8 }

  This type is for gdsl_2darray_write_func_t.

Functions

- **gdsl_2darray_t gdsl_2darray_alloc** (const char *NAME, const **ulong** R, const **ulong** C, const **gdsl_alloc_func_t** ALLOC_F, const **gdsl_free_func_t** FREE_F)

  Create a new 2D-array.

- void **gdsl_2darray_free** (**gdsl_2darray_t** A)

  Destroy a 2D-array.

- const char * **gdsl_2darray_get_name** (const **gdsl_2darray_t** A)

  Get the name of a 2D-array.

- **ulong gdsl_2darray_get_rows_number** (const **gdsl_2darray_t** A)

  Get the number of rows of a 2D-array.

- **ulong gdsl_2darray_get_columns_number** (const **gdsl_2darray_t** A)

  Get the number of columns of a 2D-array.

- **ulong gdsl_2darray_get_size** (const **gdsl_2darray_t** A)

  Get the size of a 2D-array.

- **gdsl_element_t gdsl_2darray_get_content** (const **gdsl_2darray_t** A, const **ulong** R, const **ulong** C)

  Get an element from a 2D-array.

- **gdsl_2darray_t gdsl_2darray_set_name** (**gdsl_2darray_t** A, const char *NEW_NAME)

  Set the name of a 2D-array.

- **gdsl_element_t gdsl_2darray_set_content** (**gdsl_2darray_t** A, const **ulong** R, const **ulong** C, void *VALUE)

  Modify an element in a 2D-array.

- void **gdsl_2darray_write** (const **gdsl_2darray_t** A, const **gdsl_2darray_write_func_t** WRITE_F, FILE *OUTPUT_FILE, void *USER_DATA)

  Write the content of a 2D-array to a file.

- void **gdsl_2darray_write_xml** (const **gdsl_2darray_t** A, const **gdsl_write_func_t** WRITE_F, FILE *OUTPUT_FILE, void *USER_DATA)

  Write the content of a 2D array to a file into XML.

- void **gdsl_2darray_dump** (const **gdsl_2darray_t** A, const **gdsl_write_func_t** WRITE_F, FILE *OUTPUT_FILE, void *USER_DATA)

  Dump the internal structure of a 2D array to a file.

4.7 gdsl_bstree.h File Reference

Typedefs

- typedef gdsl_bstree * **gdsl_bstree_t**

  GDSL binary search tree type.

Functions

- **gdsl_bstree_t gdsl_bstree_alloc** (const char *NAME, **gdsl_alloc_func_t** ALLOC_F, **gdsl_free_func_t** FREE_F, **gdsl_compare_func_t** COMP_F)

  Create a new binary search tree.

- void **gdsl_bstree_free** (**gdsl_bstree_t** T)

  Destroy a binary search tree.

- void **gdsl_bstree_flush** (**gdsl_bstree_t** T)

  Flush a binary search tree.

- const char * **gdsl_bstree_get_name** (const **gdsl_bstree_t** T)

  Get the name of a binary search tree.

- **bool gdsl_bstree_is_empty** (const **gdsl_bstree_t** T)

  Check if a binary search tree is empty.

- **gdsl_element_t gdsl_bstree_get_root** (const **gdsl_bstree_t** T)

  Get the root of a binary search tree.

- **ulong gdsl_bstree_get_size** (const **gdsl_bstree_t** T)

  Get the size of a binary search tree.

- **ulong gdsl_bstree_get_height** (const **gdsl_bstree_t** T)

  Get the height of a binary search tree.

- **gdsl_bstree_t gdsl_bstree_set_name** (**gdsl_bstree_t** T, const char *NEW_NAME)

  Set the name of a binary search tree.

- **gdsl_element_t gdsl_bstree_insert** (**gdsl_bstree_t** T, void *VALUE, int *RESULT)

  Insert an element into a binary search tree if it's not found or return it.

- **gdsl_element_t gdsl_bstree_remove** (**gdsl_bstree_t** T, void *VALUE)

Remove an element from a binary search tree.

- **gdsl_bstree_t gdsl_bstree_delete** (**gdsl_bstree_t** T, void *VALUE)

  Delete an element from a binary search tree.

- **gdsl_element_t gdsl_bstree_search** (const **gdsl_bstree_t** T, **gdsl_compare_func_t** COMP_F, void *VALUE)

  Search for a particular element into a binary search tree.

- **gdsl_element_t gdsl_bstree_map_prefix** (const **gdsl_bstree_t** T, **gdsl_map_func_t** MAP_F, void *USER_DATA)

  Parse a binary search tree in prefixed order.

- **gdsl_element_t gdsl_bstree_map_infix** (const **gdsl_bstree_t** T, **gdsl_map_func_t** MAP_F, void *USER_DATA)

  Parse a binary search tree in infixed order.

- **gdsl_element_t gdsl_bstree_map_postfix** (const **gdsl_bstree_t** T, **gdsl_map_func_t** MAP_F, void *USER_DATA)

  Parse a binary search tree in postfixed order.

- void **gdsl_bstree_write** (const **gdsl_bstree_t** T, **gdsl_write_func_t** WRITE_F, FILE *OUTPUT_FILE, void *USER_DATA)

  Write the element of each node of a binary search tree to a file.

- void **gdsl_bstree_write_xml** (const **gdsl_bstree_t** T, **gdsl_write_func_t** WRITE_F, FILE *OUTPUT_FILE, void *USER_DATA)

  Write the content of a binary search tree to a file into XML.

- void **gdsl_bstree_dump** (const **gdsl_bstree_t** T, **gdsl_write_func_t** WRITE_F, FILE *OUTPUT_FILE, void *USER_DATA)

  Dump the internal structure of a binary search tree to a file.

4.8 gdsl_hash.h File Reference

Typedefs

- typedef hash_table * **gdsl_hash_t**
  GDSL hashtable type.
- typedef const char *(* **gdsl_key_func_t**)(void *VALUE)
  GDSL hashtable key function type.
- typedef const **ulong**(* **gdsl_hash_func_t**)(const char *KEY)
  GDSL hashtable hash function type.

Functions

- const **ulong gdsl_hash** (const char *KEY)
  Computes a hash value from a NULL terminated character string.
- **gdsl_hash_t gdsl_hash_alloc** (const char *NAME, **gdsl_alloc_func_t** ALLOC_F, **gdsl_free_func_t** FREE_F, **gdsl_key_func_t** KEY_F, **gdsl_hash_func_t** HASH_F, ushort INITIAL_ENTRIES_NB)
  Create a new hashtable.
- void **gdsl_hash_free** (**gdsl_hash_t** H)
  Destroy a hashtable.
- void **gdsl_hash_flush** (**gdsl_hash_t** H)
  Flush a hashtable.
- const char * **gdsl_hash_get_name** (const **gdsl_hash_t** H)
  Get the name of a hashtable.
- ushort **gdsl_hash_get_entries_number** (const **gdsl_hash_t** H)
  Get the number of entries of a hashtable.
- ushort **gdsl_hash_get_lists_max_size** (const **gdsl_hash_t** H)
  Get the max number of elements allowed in each entry of a hashtable.
- ushort **gdsl_hash_get_longest_list_size** (const **gdsl_hash_t** H)
  Get the number of elements of the longest list entry of a hashtable.
- **ulong gdsl_hash_get_size** (const **gdsl_hash_t** H)
  Get the size of a hashtable.

- double **gdsl_hash_get_fill_factor** (const **gdsl_hash_t** H)

  Get the fill factor of a hashtable.

- **gdsl_hash_t gdsl_hash_set_name** (**gdsl_hash_t** H, const char *NEW_NAME)

  Set the name of a hashtable.

- **gdsl_element_t gdsl_hash_insert** (**gdsl_hash_t** H, void *VALUE)

  Insert an element into a hashtable (PUSH).

- **gdsl_element_t gdsl_hash_remove** (**gdsl_hash_t** H, const char *KEY)

  Remove an element from a hashtable (POP).

- **gdsl_hash_t gdsl_hash_delete** (**gdsl_hash_t** H, const char *KEY)

  Delete an element from a hashtable.

- **gdsl_hash_t gdsl_hash_modify** (**gdsl_hash_t** H, ushort NEW_ENTRIES_NB, ushort NEW_LISTS_MAX_SIZE)

  Increase the dimensions of a hashtable.

- **gdsl_element_t gdsl_hash_search** (const **gdsl_hash_t** H, const char *KEY)

  Search for a particular element into a hashtable (GET).

- **gdsl_element_t gdsl_hash_map** (const **gdsl_hash_t** H, **gdsl_map_func_t** MAP_F, void *USER_DATA)

  Parse a hashtable.

- void **gdsl_hash_write** (const **gdsl_hash_t** H, **gdsl_write_func_t** WRITE_F, FILE *OUTPUT_FILE, void *USER_DATA)

  Write all the elements of a hashtable to a file.

- void **gdsl_hash_write_xml** (const **gdsl_hash_t** H, **gdsl_write_func_t** WRITE_F, FILE *OUTPUT_FILE, void *USER_DATA)

  Write the content of a hashtable to a file into XML.

- void **gdsl_hash_dump** (const **gdsl_hash_t** H, **gdsl_write_func_t** WRITE_F, FILE *OUTPUT_FILE, void *USER_DATA)

  Dump the internal structure of a hashtable to a file.

4.9 gdsl_list.h File Reference

Typedefs

- typedef _gdsl_list * **gdsl_list_t**
  GDSL doubly-linked list type.
- typedef _gdsl_list_cursor * **gdsl_list_cursor_t**
  GDSL doubly-linked list cursor type.

Functions

- **gdsl_list_t gdsl_list_alloc** (const char *NAME, **gdsl_alloc_func_t** ALLOC_F, **gdsl_free_func_t** FREE_F)
  Create a new list.
- void **gdsl_list_free** (**gdsl_list_t** L)
  Destroy a list.
- void **gdsl_list_flush** (**gdsl_list_t** L)
  Flush a list.
- const char * **gdsl_list_get_name** (const **gdsl_list_t** L)
  Get the name of a list.
- **ulong gdsl_list_get_size** (const **gdsl_list_t** L)
  Get the size of a list.
- **bool gdsl_list_is_empty** (const **gdsl_list_t** L)
  Check if a list is empty.
- **gdsl_element_t gdsl_list_get_head** (const **gdsl_list_t** L)
  Get the head of a list.
- **gdsl_element_t gdsl_list_get_tail** (const **gdsl_list_t** L)
  Get the tail of a list.
- **gdsl_list_t gdsl_list_set_name** (**gdsl_list_t** L, const char *NEW_NAME)
  Set the name of a list.
- **gdsl_element_t gdsl_list_insert_head** (**gdsl_list_t** L, void *VALUE)
  Insert an element at the head of a list.

- **gdsl_element_t gdsl_list_insert_tail** (**gdsl_list_t** L, void *VALUE)

  Insert an element at the tail of a list.

- **gdsl_element_t gdsl_list_remove_head** (**gdsl_list_t** L)

  Remove the head of a list.

- **gdsl_element_t gdsl_list_remove_tail** (**gdsl_list_t** L)

  Remove the tail of a list.

- **gdsl_element_t gdsl_list_remove** (**gdsl_list_t** L, **gdsl_compare_func_t** COMP_F, const void *VALUE)

  Remove a particular element from a list.

- **gdsl_list_t gdsl_list_delete_head** (**gdsl_list_t** L)

  Delete the head of a list.

- **gdsl_list_t gdsl_list_delete_tail** (**gdsl_list_t** L)

  Delete the tail of a list.

- **gdsl_list_t gdsl_list_delete** (**gdsl_list_t** L, **gdsl_compare_func_t** COMP_F, const void *VALUE)

  Delete a particular element from a list.

- **gdsl_element_t gdsl_list_search** (const **gdsl_list_t** L, **gdsl_compare_func_t** COMP_F, const void *VALUE)

  Search for a particular element into a list.

- **gdsl_element_t gdsl_list_search_by_position** (const **gdsl_list_t** L, **ulong** POS)

  Search for an element by its position in a list.

- **gdsl_element_t gdsl_list_search_max** (const **gdsl_list_t** L, **gdsl_compare_func_t** COMP_F)

  Search for the greatest element of a list.

- **gdsl_element_t gdsl_list_search_min** (const **gdsl_list_t** L, **gdsl_compare_func_t** COMP_F)

  Search for the lowest element of a list.

- **gdsl_list_t gdsl_list_sort** (**gdsl_list_t** L, **gdsl_compare_func_t** COMP_F, **gdsl_element_t** MAX)

  Sort a list.

- **gdsl_element_t gdsl_list_map_forward** (const **gdsl_list_t** L, **gdsl_map_func_t** MAP_F, void *USER_DATA)

  Parse a list from head to tail.

- **gdsl_element_t gdsl_list_map_backward** (const **gdsl_list_t** L, **gdsl_map_func_t** MAP_F, void *USER_DATA)

  Parse a list from tail to head.

- void **gdsl_list_write** (const **gdsl_list_t** L, **gdsl_write_func_t** WRITE_F, FILE *OUTPUT_FILE, void *USER_DATA)

  Write all the elements of a list to a file.

- void **gdsl_list_write_xml** (const **gdsl_list_t** L, **gdsl_write_func_t** WRITE_F, FILE *OUTPUT_FILE, void *USER_DATA)

  Write the content of a list to a file into XML.

- void **gdsl_list_dump** (const **gdsl_list_t** L, **gdsl_write_func_t** WRITE_F, FILE *OUTPUT_FILE, void *USER_DATA)

  Dump the internal structure of a list to a file.

- **gdsl_list_cursor_t gdsl_list_cursor_alloc** (const **gdsl_list_t** L)

  Create a new list cursor.

- void **gdsl_list_cursor_free** (**gdsl_list_cursor_t** C)

  Destroy a list cursor.

- void **gdsl_list_cursor_move_to_head** (**gdsl_list_cursor_t** C)

  Put a cursor on the head of its list.

- void **gdsl_list_cursor_move_to_tail** (**gdsl_list_cursor_t** C)

  Put a cursor on the tail of its list.

- **gdsl_element_t gdsl_list_cursor_move_to_value** (**gdsl_list_cursor_t** C, **gdsl_compare_func_t** COMP_F, void *VALUE)

  Place a cursor on a particular element.

- **gdsl_element_t gdsl_list_cursor_move_to_position** (**gdsl_list_cursor_t** C, **ulong** POS)

  Place a cursor on a element given by its position.

- void **gdsl_list_cursor_step_forward** (**gdsl_list_cursor_t** C)

  Move a cursor one step forward of its list.

- void **gdsl_list_cursor_step_backward** (**gdsl_list_cursor_t** C)

  Move a cursor one step backward of its list.

- **bool gdsl_list_cursor_is_on_head** (const **gdsl_list_cursor_t** C)

  Check if a cursor is on the head of its list.

- **bool gdsl_list_cursor_is_on_tail** (const **gdsl_list_cursor_t** C)

  Check if a cursor is on the tail of its list.

- **bool gdsl_list_cursor_has_succ** (const **gdsl_list_cursor_t** C)

  Check if a cursor has a successor.

- **bool gdsl_list_cursor_has_pred** (const **gdsl_list_cursor_t** C)

  Check if a cursor has a predecessor.

- void **gdsl_list_cursor_set_content** (**gdsl_list_cursor_t** C, **gdsl_element_t** E)

  Set the content of the cursor.

- **gdsl_element_t gdsl_list_cursor_get_content** (const **gdsl_list_cursor_t** C)

  Get the content of a cursor.

- **gdsl_element_t gdsl_list_cursor_insert_after** (**gdsl_list_cursor_t** C, void *VALUE)

  Insert a new element after a cursor.

- **gdsl_element_t gdsl_list_cursor_insert_before** (**gdsl_list_cursor_t** C, void *VALUE)

  Insert a new element before a cursor.

- **gdsl_element_t gdsl_list_cursor_remove** (**gdsl_list_cursor_t** C)

  Removec the element under a cursor.

- **gdsl_element_t gdsl_list_cursor_remove_after** (**gdsl_list_cursor_t** C)

  Removec the element after a cursor.

- **gdsl_element_t gdsl_list_cursor_remove_before** (**gdsl_list_cursor_t** C)

  Remove the element before a cursor.

- **gdsl_list_cursor_t gdsl_list_cursor_delete** (**gdsl_list_cursor_t** C)

  Delete the element under a cursor.

- **gdsl_list_cursor_t gdsl_list_cursor_delete_after** (**gdsl_list_cursor_t** C)

  Delete the element after a cursor.

- **gdsl_list_cursor_t gdsl_list_cursor_delete_before (gdsl_list_cursor_t C)**

  Delete the element before the cursor of a list.

4.10 gdsl_macros.h File Reference

Defines

- #define **GDSL_MAX**(X, Y) (X>Y?X:Y)
  Give the greatest number of two numbers.
- #define **GDSL_MIN**(X, Y) (X>Y?Y:X)
  Give the lowest number of two numbers.

4.11 gdsl_perm.h File Reference

Typedefs

- typedef gdsl_perm * **gdsl_perm_t**

  GDSL permutation type.

- typedef void(* **gdsl_perm_write_func_t**)(**ulong** E, FILE *OUTPUT_FILE, **gdsl_perm_position_t** POSITION, void *USER_DATA)

  GDSL permutation write function type.

Enumerations

- enum **gdsl_perm_position_t** { **GDSL_PERM_POSITION_FIRST** = 1, **GDSL_PERM_POSITION_LAST** = 2 }

  This type is for gdsl_perm_write_func_t.

Functions

- **gdsl_perm_t gdsl_perm_alloc** (const char *NAME, const **ulong** N)

  Create a new permutation.

- void **gdsl_perm_free** (**gdsl_perm_t** P)

  Destroy a permutation.

- **gdsl_perm_t gdsl_perm_copy** (const **gdsl_perm_t** P)

  Copy a permutation.

- const char * **gdsl_perm_get_name** (const **gdsl_perm_t** P)

  Get the name of a permutation.

- **ulong gdsl_perm_get_size** (const **gdsl_perm_t** P)

  Get the size of a permutation.

- **ulong gdsl_perm_get_element** (const **gdsl_perm_t** P, const **ulong** INDIX)

  Get the (INDIX+1)-th element from a permutation.

- **ulong** * **gdsl_perm_get_elements_array** (const **gdsl_perm_t** P)

  Get the array elements of a permutation.

- **ulong gdsl_perm_linear_inversions_count** (const **gdsl_perm_t** P)

  Count the inversions number into a linear permutation.

- **ulong gdsl_perm_linear_cycles_count** (const **gdsl_perm_t** P)

  Count the cycles number into a linear permutation.

- **ulong gdsl_perm_canonical_cycles_count** (const **gdsl_perm_t** P)

  Count the cycles number into a canonical permutation.

- **gdsl_perm_t gdsl_perm_set_name** (**gdsl_perm_t** P, const char *NEW_NAME)

  Set the name of a permutation.

- **gdsl_perm_t gdsl_perm_linear_next** (**gdsl_perm_t** P)

  Get the next permutation from a linear permutation.

- **gdsl_perm_t gdsl_perm_linear_prev** (**gdsl_perm_t** P)

  Get the previous permutation from a linear permutation.

- **gdsl_perm_t gdsl_perm_set_elements_array** (**gdsl_perm_t** P, const **ulong** *ARRAY)

  Initialize a permutation with an array of values.

- **gdsl_perm_t gdsl_perm_multiply** (**gdsl_perm_t** RESULT, const **gdsl_perm_t** ALPHA, const **gdsl_perm_t** BETA)

  Multiply two permutations.

- **gdsl_perm_t gdsl_perm_linear_to_canonical** (**gdsl_perm_t** Q, const **gdsl_perm_t** P)

  Convert a linear permutation to its canonical form.

- **gdsl_perm_t gdsl_perm_canonical_to_linear** (**gdsl_perm_t** Q, const **gdsl_perm_t** P)

  Convert a canonical permutation to its linear form.

- **gdsl_perm_t gdsl_perm_inverse** (**gdsl_perm_t** P)

  Inverse in place a permutation.

- **gdsl_perm_t gdsl_perm_reverse** (**gdsl_perm_t** P)

  Reverse in place a permutation.

- **gdsl_perm_t gdsl_perm_randomize** (**gdsl_perm_t** P)

  Randomize a permutation.

- **gdsl_element_t * gdsl_perm_apply_on_array** (**gdsl_element_t** *V, const **gdsl_perm_t** P)

  Apply a permutation on to a vector.

- void **gdsl_perm_write** (const **gdsl_perm_t** P, const **gdsl_perm_write_func_t** WRITE_F, FILE *OUTPUT_FILE, void *USER_DATA)

  Write the elements of a permutation to a file.

- void **gdsl_perm_write_xml** (const **gdsl_perm_t** P, const **gdsl_write_func_t** WRITE_F, FILE *OUTPUT_FILE, void *USER_DATA)

  Write the elements of a permutation to a file into XML.

- void **gdsl_perm_dump** (const **gdsl_perm_t** P, const **gdsl_write_func_t** WRITE_F, FILE *OUTPUT_FILE, void *USER_DATA)

  Dump the internal structure of a permutation to a file.

4.12 gdsl_queue.h File Reference

Typedefs

- typedef _gdsl_queue * **gdsl_queue_t**

  GDSL queue type.

Functions

- **gdsl_queue_t gdsl_queue_alloc** (const char *NAME, **gdsl_alloc_func_t** ALLOC_F, **gdsl_free_func_t** FREE_F)

  Create a new queue.

- void **gdsl_queue_free** (**gdsl_queue_t** Q)

  Destroy a queue.

- void **gdsl_queue_flush** (**gdsl_queue_t** Q)

  Flush a queue.

- const char * **gdsl_queue_get_name** (const **gdsl_queue_t** Q)

  Getsthe name of a queue.

- **ulong gdsl_queue_get_size** (const **gdsl_queue_t** Q)

  Get the size of a queue.

- **bool gdsl_queue_is_empty** (const **gdsl_queue_t** Q)

  Check if a queue is empty.

- **gdsl_element_t gdsl_queue_get_head** (const **gdsl_queue_t** Q)

  Get the head of a queue.

- **gdsl_element_t gdsl_queue_get_tail** (const **gdsl_queue_t** Q)

  Get the tail of a queue.

- **gdsl_queue_t gdsl_queue_set_name** (**gdsl_queue_t** Q, const char *NEW_NAME)

  Set the name of a queue.

- **gdsl_element_t gdsl_queue_insert** (**gdsl_queue_t** Q, void *VALUE)

  Insert an element in a queue (PUT).

- **gdsl_element_t gdsl_queue_remove** (**gdsl_queue_t** Q)

  Remove an element from a queue (GET).

- **gdsl_element_t gdsl_queue_search** (const **gdsl_queue_t** Q, **gdsl_compare_func_t** COMP_F, void *VALUE)

  Search for a particular element in a queue.

- **gdsl_element_t gdsl_queue_search_by_position** (const **gdsl_queue_t** Q, **ulong** POS)

  Search for an element by its position in a queue.

- **gdsl_element_t gdsl_queue_map_forward** (const **gdsl_queue_t** Q, **gdsl_map_func_t** MAP_F, void *USER_DATA)

  Parse a queue from head to tail.

- **gdsl_element_t gdsl_queue_map_backward** (const **gdsl_queue_t** Q, **gdsl_map_func_t** MAP_F, void *USER_DATA)

  Parse a queue from tail to head.

- void **gdsl_queue_write** (const **gdsl_queue_t** Q, **gdsl_write_func_t** WRITE_F, FILE *OUTPUT_FILE, void *USER_DATA)

  Write all the elements of a queue to a file.

- void **gdsl_queue_write_xml** (const **gdsl_queue_t** Q, **gdsl_write_func_t** WRITE_F, FILE *OUTPUT_FILE, void *USER_DATA)

  Write the content of a queue to a file into XML.

- void **gdsl_queue_dump** (const **gdsl_queue_t** Q, **gdsl_write_func_t** WRITE_F, FILE *OUTPUT_FILE, void *USER_DATA)

  Dump the internal structure of a queue to a file.

4.13 gdsl_rbtree.h File Reference

Typedefs

- typedef gdsl_rbtree * **gdsl_rbtree_t**

Functions

- **gdsl_rbtree_t gdsl_rbtree_alloc** (const char *NAME, **gdsl_alloc_func_t** ALLOC_F, **gdsl_free_func_t** FREE_F, **gdsl_compare_func_t** COMP_F)

  Create a new red-black tree.

- void **gdsl_rbtree_free** (**gdsl_rbtree_t** T)

  Destroy a red-black tree.

- void **gdsl_rbtree_flush** (**gdsl_rbtree_t** T)

  Flush a red-black tree.

- char * **gdsl_rbtree_get_name** (const **gdsl_rbtree_t** T)

  Get the name of a red-black tree.

- **bool gdsl_rbtree_is_empty** (const **gdsl_rbtree_t** T)

  Check if a red-black tree is empty.

- **gdsl_element_t gdsl_rbtree_get_root** (const **gdsl_rbtree_t** T)

  Get the root of a red-black tree.

- **ulong gdsl_rbtree_get_size** (const **gdsl_rbtree_t** T)

  Get the size of a red-black tree.

- **ulong gdsl_rbtree_height** (const **gdsl_rbtree_t** T)

  Get the height of a red-black tree.

- **gdsl_rbtree_t gdsl_rbtree_set_name** (**gdsl_rbtree_t** T, const char *NEW_NAME)

  Set the name of a red-black tree.

- **gdsl_element_t gdsl_rbtree_insert** (**gdsl_rbtree_t** T, void *VALUE, int *RESULT)

  Insert an element into a red-black tree if it's not found or return it.

- **gdsl_element_t gdsl_rbtree_remove** (**gdsl_rbtree_t** T, void *VALUE)

  Remove an element from a red-black tree.

- **gdsl_rbtree_t gdsl_rbtree_delete** (**gdsl_rbtree_t** T, void *VALUE)

  Delete an element from a red-black tree.

- **gdsl_element_t gdsl_rbtree_search** (const **gdsl_rbtree_t** T, **gdsl_compare_func_t** COMP_F, void *VALUE)

  Search for a particular element into a red-black tree.

- **gdsl_element_t gdsl_rbtree_map_prefix** (const **gdsl_rbtree_t** T, **gdsl_map_func_t** MAP_F, void *USER_DATA)

  Parse a red-black tree in prefixed order.

- **gdsl_element_t gdsl_rbtree_map_infix** (const **gdsl_rbtree_t** T, **gdsl_map_func_t** MAP_F, void *USER_DATA)

  Parse a red-black tree in infixed order.

- **gdsl_element_t gdsl_rbtree_map_postfix** (const **gdsl_rbtree_t** T, **gdsl_map_func_t** MAP_F, void *USER_DATA)

  Parse a red-black tree in postfixed order.

- void **gdsl_rbtree_write** (const **gdsl_rbtree_t** T, **gdsl_write_func_t** WRITE_F, FILE *OUTPUT_FILE, void *USER_DATA)

  Write the element of each node of a red-black tree to a file.

- void **gdsl_rbtree_write_xml** (const **gdsl_rbtree_t** T, **gdsl_write_func_t** WRITE_F, FILE *OUTPUT_FILE, void *USER_DATA)

  Write the content of a red-black tree to a file into XML.

- void **gdsl_rbtree_dump** (const **gdsl_rbtree_t** T, **gdsl_write_func_t** WRITE_F, FILE *OUTPUT_FILE, void *USER_DATA)

  Dump the internal structure of a red-black tree to a file.

4.14 gdsl_sort.h File Reference

Functions

- void **gdsl_sort** (**gdsl_element_t** *T, **ulong** N, **gdsl_compare_-func_t** COMP_F)

  Sort an array in place.

4.15 gdsl_stack.h File Reference

Typedefs

- typedef _gdsl_stack * **gdsl_stack_t**
  GDSL stack type.

Functions

- **gdsl_stack_t gdsl_stack_alloc** (const char *NAME, **gdsl_alloc_-func_t** ALLOC_F, **gdsl_free_func_t** FREE_F)
  Create a new stack.
- void **gdsl_stack_free** (**gdsl_stack_t** S)
  Destroy a stack.
- void **gdsl_stack_flush** (**gdsl_stack_t** S)
  Flush a stack.
- const char * **gdsl_stack_get_name** (const **gdsl_stack_t** S)
  Getsthe name of a stack.
- **ulong gdsl_stack_get_size** (const **gdsl_stack_t** S)
  Get the size of a stack.
- ubyte **gdsl_stack_get_growing_factor** (const **gdsl_stack_t** S)
  Get the growing factor of a stack.
- **bool gdsl_stack_is_empty** (const **gdsl_stack_t** S)
  Check if a stack is empty.
- **gdsl_element_t gdsl_stack_get_top** (const **gdsl_stack_t** S)
  Get the top of a stack.
- **gdsl_element_t gdsl_stack_get_bottom** (const **gdsl_stack_t** S)
  Get the bottom of a stack.
- **gdsl_stack_t gdsl_stack_set_name** (**gdsl_stack_t** S, const char *NEW_NAME)
  Set the name of a stack.
- void **gdsl_stack_set_growing_factor** (**gdsl_stack_t** S, ubyte G)
  Set the growing factor of a stack.

- **gdsl_element_t gdsl_stack_insert** (**gdsl_stack_t** S, void *VALUE)

  Insert an element in a stack (PUSH).

- **gdsl_element_t gdsl_stack_remove** (**gdsl_stack_t** S)

  Remove an element from a stack (POP).

- **gdsl_element_t gdsl_stack_search** (const **gdsl_stack_t** S, **gdsl_compare_func_t** COMP_F, void *VALUE)

  Search for a particular element in a stack.

- **gdsl_element_t gdsl_stack_search_by_position** (const **gdsl_stack_t** S, **ulong** POS)

  Search for an element by its position in a stack.

- **gdsl_element_t gdsl_stack_map_forward** (const **gdsl_stack_t** S, **gdsl_map_func_t** MAP_F, void *USER_DATA)

  Parse a stack from bottom to top.

- **gdsl_element_t gdsl_stack_map_backward** (const **gdsl_stack_t** S, **gdsl_map_func_t** MAP_F, void *USER_DATA)

  Parse a stack from top to bottom.

- void **gdsl_stack_write** (const **gdsl_stack_t** S, **gdsl_write_func_t** WRITE_F, FILE *OUTPUT_FILE, void *USER_DATA)

  Write all the elements of a stack to a file.

- void **gdsl_stack_write_xml** (**gdsl_stack_t** S, **gdsl_write_func_t** WRITE_F, FILE *OUTPUT_FILE, void *USER_DATA)

  Write the content of a stack to a file into XML.

- void **gdsl_stack_dump** (**gdsl_stack_t** S, **gdsl_write_func_t** WRITE_F, FILE *OUTPUT_FILE, void *USER_DATA)

  Dump the internal structure of a stack to a file.

4.16 gdsl_types.h File Reference

Typedefs

- typedef void * **gdsl_element_t**

  GDSL element type.

- typedef **gdsl_element_t**(* **gdsl_alloc_func_t**)(void *USER_DATA)

  GDSL Alloc element function type.

- typedef void(* **gdsl_free_func_t**)(**gdsl_element_t** E)

  GDSL Free element function type.

- typedef **gdsl_element_t**(* **gdsl_copy_func_t**)(const **gdsl_element_t** E)

  GDSL Copy element function type.

- typedef int(* **gdsl_map_func_t**)(const **gdsl_element_t** E, void *USER_DATA)

  GDSL Map element function type.

- typedef int(* **gdsl_compare_func_t**)(const **gdsl_element_t** E, void *VALUE)

  GDSL Comparison element function type.

- typedef void(* **gdsl_write_func_t**)(const **gdsl_element_t** E, FILE *OUTPUT_FILE, void *USER_DATA)

  GDSL Write element function type.

- typedef unsigned long int **ulong**

Enumerations

- enum **gdsl_constant_t** {

  GDSL_ERR_MEM_ALLOC = -1, **GDSL_MAP_STOP** = 0, **GDSL_MAP_CONT** = 1, **GDSL_INSERTED**,

  GDSL_FOUND }

  GDSL Constants.

- enum **bool** { **FALSE** = 0, **TRUE** = 1 }

Index