gdsl

1.8

Generated by Doxygen 1.7.6.1

Tue Apr 3 2018 15:15:50

Contents

Chapter 1

gdsl

1.1 Introduction

The Generic Data Structures Library (GDSL) is a collection of routines for generic data structures manipulation. It is a portable and re-entrant library fully written from scratch in pure ANSI C. It is designed to offer for C programmers common data structures with powerful algorithms, and hidden implementation. Available structures are lists, queues, stacks, hash-tables, binary trees, search binary trees, red-black trees, 2D arrays, permutations, heaps and interval heaps.

1.2 About

This is the gdsl (Release 1.8) documentation.

1.3 Copyright

1.4 Project Manager

Nicolas Darnis <ndarnis@free.fr>.

1.5 Authors

Nicolas Darnis <ndarnis@free.fr>: all GDSL modules excepted the ones listed below.

Peter Kerpedjiev <pkerpedjiev@gmail.com>: interval_heap module.

1.6 Thanks

This is the list of persons (in randomized order) the GDSL Team want to thanks for their direct and/or indirect help:

- Vincent Vidal <vidal@cril.univ-artois.fr>

For his bug report in hash_insert method and into **gdsl.h** (p. 261).

- Martin Pichlmair <pi@igw.tuwien.ac.at>

For his patch to compile GDSL under OSX.

- Mathieu Clabaut <mathieu.clabaut@gmail.com>

For his bug report in **gdsl_stack_insert()** (p. 238).

- Xavier De Labouret <Xavier.de_Labouret@cvf.fr>

For his bug report in **gdsl_hash_search()** (p. 110).

- Kaz Kylheku <kaz@ashi.footprints.net>

For his KazLib from wich the deletion algorithm for gdsl_rbtree.c is inspired.

- David Lewin <dlewin@free.fr>

For his bug report in **gdsl_list_map_backward()** (p. 160), and for the problem of redefining bool type in **gdsl_types.h** (p. 283).

- Torsten Luettgert <t.luettgert@combox.de>

For his gdsl.spec file to build GDSL's RPM package.

- Charles F. Randall <cfriv@yahoo.com>

For his patch to compile GDSL under FreeBSD.

- Sascha Alexander Jopen <jopen@informatik.uni-bonn.de>

For his patch to compile GDSL under Android OS.

- Peter Kerpedjiev <pkerpedjiev@gmail.com>

For his gdsl_interval_heap module.

- Benny Pasternak <bennypk>

For his is bug report in gdsl_rbtree_map_infix function.

The GDSL Team.

Chapter 2

Module Index

2.1 Modules

Here is a list of all modules:

Chapter 3

File Index

3.1 File List

Here is a list of all files with brief descriptions:

Chapter 4

Module Documentation

4.1 Low level binary tree manipulation module.

This module is for manipulation of low-level binary trees.

Typedefs

- typedef struct _gdsl_bintree ∗ **_gdsl_bintree_t**

  GDSL low-level binary tree type.
- typedef int(∗ **_gdsl_bintree_map_func_t**)(const **_gdsl_bintree_t** TREE, void ∗USER_DATA)

  GDSL low-level binary tree map function type.
- typedef void(∗ **_gdsl_bintree_write_func_t**)(const **_gdsl_bintree_t** TREE, FILE ∗OUTPUT_FILE, void ∗USER_DATA)

  GDSL low-level binary tree write 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_bintree_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_bintree-_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_bintree_-write_func_t** WRITE_F, FILE ∗OUTPUT_FILE, void ∗USER_DATA)

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

4.1.1 Detailed Description

This module is for manipulation of low-level binary trees.

4.1.2 Copyright

4.1.3 Typedef Documentation

4.1.3.1 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 57 of file _gdsl_bintree.h.

4.1.3.2 typedef int(∗ **_gdsl_bintree_map_func_t**)(const **_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

GDSL_MAP_STOP if the mapping must be stopped.
GDSL_MAP_CONT if the mapping must be continued.

Definition at line 66 of file _gdsl_bintree.h.

4.1.3.3 typedef void(∗ _gdsl_bintree_write_func_t)(const _gdsl_bintree_t TREE, FILE ∗OUTPUT_FILE, void ∗USER_DATA)

GDSL low-level binary tree write function type.

Parameters

TREE	The low-level binary tree to write.
OUTPUT_FILE	The file where to write TREE.
USER_DATA	The user datas to pass to this function.

Definition at line 76 of file _gdsl_bintree.h.

4.1.4 Function Documentation

4.1.4.1 _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. 13)

Examples:

examples/main_llbintree.c.

4.1.4.2 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. 12)

Examples:

examples/main_llbintree.c.

4.1.4.3 _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. 12)
_gdsl_bintree_free() (p. 13)
_gdsl_bintree_is_empty() (p. 14)

Examples:

examples/main_llbintree.c.

4.1.4.4 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)

4.1.4.5 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)

4.1.4.6 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)

4.1.4.7 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. 21)

Examples:

examples/main_llbintree.c.

4.1.4.8 _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. 21)

4.1.4.9 _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. 18)
_gdsl_bintree_set_left() (p. 22)
_gdsl_bintree_set_right() (p. 22)

4.1.4.10 _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. 17)
_gdsl_bintree_set_left() (p. 22)
_gdsl_bintree_set_right() (p. 22)

4.1.4.11 _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. 19)

4.1.4.12 _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. 18)

4.1.4.13 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. 20)

4.1.4.14 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. 19)

4.1.4.15 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. 16)

4.1.4.16 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. 17)

4.1.4.17 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. 22)
_gdsl_bintree_get_left() (p. 17)
_gdsl_bintree_get_right() (p. 18)

4.1.4.18 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. 22)
_gdsl_bintree_get_left() (p. 17)
_gdsl_bintree_get_right() (p. 18)

4.1.4.19 _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. 23)
_gdsl_bintree_rotate_left_right() (p. 24)
_gdsl_bintree_rotate_right_left() (p. 25)

Examples:

examples/main_llbintree.c.

4.1.4.20 _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. 23)
_gdsl_bintree_rotate_left_right() (p. 24)
_gdsl_bintree_rotate_right_left() (p. 25)

Examples:

examples/main_llbintree.c.

4.1.4.21 _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. 23)
_gdsl_bintree_rotate_right() (p. 23)
_gdsl_bintree_rotate_right_left() (p. 25)

4.1.4.22 _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. 23)
_gdsl_bintree_rotate_right() (p. 23)
_gdsl_bintree_rotate_left_right() (p. 24)

4.1.4.23 _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. 25) 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. 26)
_gdsl_bintree_map_postfix() (p. 27)

Examples:

examples/main_llbintree.c.

4.1.4.24 _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. 26) 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. 25)
_gdsl_bintree_map_postfix() (p. 27)

Examples:

examples/main_llbintree.c.

4.1.4.25 _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. 27) 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. 25)
_gdsl_bintree_map_infix() (p. 26)

Examples:

examples/main_llbintree.c.

4.1.4.26 void _gdsl_bintree_write (const _gdsl_bintree_t *T*, const _gdsl_bintree_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. 28)
_gdsl_bintree_dump() (p. 29)

4.1.4.27 void _gdsl_bintree_write_xml (const _gdsl_bintree_t *T*, const _gdsl_bintree_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_F-ILE	The file where to write T's nodes.
USER_DAT-A	User's datas passed to WRITE_F.

See also

_gdsl_bintree_write() (p. 28)
_gdsl_bintree_dump() (p. 29)

Examples:

examples/main_llbintree.c.

4.1.4.28 void _gdsl_bintree_dump (const _gdsl_bintree_t *T*, const _gdsl_bintree_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_F-ILE	The file where to write T's nodes.
USER_DAT-A	User's datas passed to WRITE_F.

See also

_gdsl_bintree_write() (p. 28)
_gdsl_bintree_write_xml() (p. 28)

Examples:

examples/main_llbintree.c.

4.2 Low-level binary search tree manipulation module.

This module is for manipulation of low-level binary search trees.

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.
- typedef void(∗ **_gdsl_bstree_write_func_t**)(**_gdsl_bstree_t** TREE, FILE ∗OU-TPUT_FILE, void ∗USER_DATA)

  GDSL low-level binary search tree write 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_bstree_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_bstree_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_bstree_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.

4.2.1 Detailed Description

This module is for manipulation of low-level binary search trees.

4.2.2 Copyright

4.2.3 Typedef Documentation

4.2.3.1 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 55 of file _gdsl_bstree.h.

4.2.3.2 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 map.
USER_DATA	The user datas to pass to this function.

Returns

GDSL_MAP_STOP if the mapping must be stopped.
GDSL_MAP_CONT if the mapping must be continued.

Definition at line 64 of file _gdsl_bstree.h.

4.2.3.3 typedef void(∗ _gdsl_bstree_write_func_t)(_gdsl_bstree_t TREE, FILE ∗OUTPUT_FILE, void ∗USER_DATA)

GDSL low-level binary search tree write function type.

Parameters

TREE	The low-level binary search tree to write.
OUTPUT_FILE	The file where to write TREE.
USER_DATA	The user datas to pass to this function.

Definition at line 74 of file _gdsl_bstree.h.

4.2.4 Function Documentation

4.2.4.1 _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. 34)

Examples:

examples/main_llbstree.c.

4.2.4.2 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. 34)

Examples:

examples/main_llbstree.c.

4.2.4.3 _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. 34)
_gdsl_bstree_free() (p. 34)
_gdsl_bstree_is_empty() (p. 36)

4.2.4.4 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. 36)
_gdsl_bstree_is_root() (p. 37)

4.2.4.5 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. 36)
_gdsl_bstree_is_root() (p. 37)

4.2.4.6 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.

Examples:

examples/main_llbstree.c.

4.2.4.7 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. 36)
_gdsl_bstree_is_leaf() (p. 36)

4.2.4.8 _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. 37)

4.2.4.9 _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. 39)

4.2.4.10 _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. 38)

4.2.4.11 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. 40)

4.2.4.12 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. 39)

4.2.4.13 _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. 42)
_gdsl_bstree_remove() (p. 41)

Examples:

examples/main_llbstree.c.

4.2.4.14 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 V-ALUE, 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. 40)
_gdsl_bstree_search() (p. 42)

4.2.4.15 _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. 40)
_gdsl_bstree_remove() (p. 41)

4.2.4.16 _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.

4.2.4.17 _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. 43) 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. 44)
_gdsl_bstree_map_postfix() (p. 45)

4.2.4.18 _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. 44) 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. 43)
_gdsl_bstree_map_postfix() (p. 45)

4.2.4.19 _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. 45) 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. 43)
_gdsl_bstree_map_infix() (p. 44)

4.2.4.20 void _gdsl_bstree_write (const _gdsl_bstree_t *T*, const _gdsl_bstree_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. 46)
_gdsl_bstree_dump() (p. 47)

4.2.4.21 void _gdsl_bstree_write_xml (const _gdsl_bstree_t *T*, const _gdsl_bstree_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 W-RITE_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. 46)
_gdsl_bstree_dump() (p. 47)

Examples:

examples/main_llbstree.c.

4.2.4.22 void _gdsl_bstree_dump (const _gdsl_bstree_t *T*, const _gdsl_bstree_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. 46)
_gdsl_bstree_write_xml() (p. 46)

4.3 Low-level doubly-linked list manipulation module.

This module is for manipulation of low-level doubly-linked lists.

Typedefs

- typedef **_gdsl_node_t _gdsl_list_t**
  GDSL low-level doubly-linked list 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_node-_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_-node_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_node_write_func_t** - WRITE_F, FILE ∗OUTPUT_FILE, void ∗USER_DATA)
  Write all nodes of a low-level list to a file.
- void **_gdsl_list_write_xml** (const **_gdsl_list_t** L, const **_gdsl_node_write_-func_t** WRITE_F, FILE ∗OUTPUT_FILE, void ∗USER_DATA)
  Write all nodes of a low-level list to a file into XML.
- void **_gdsl_list_dump** (const **_gdsl_list_t** L, const **_gdsl_node_write_func_t** WRITE_F, FILE ∗OUTPUT_FILE, void ∗USER_DATA)
  Dump the internal structure of a low-level list to a file.

4.3.1 Detailed Description

This module is for manipulation of low-level doubly-linked lists.

4.3.2 Copyright

4.3.3 Typedef Documentation

4.3.3.1 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 57 of file _gdsl_list.h.

4.3.4 Function Documentation

4.3.4.1 _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. 50)

Examples:

examples/main_lllist.c.

4.3.4.2 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. 49)

Examples:

examples/main_lllist.c.

4.3.4.3 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.

4.3.4.4 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 $|L|$).

Examples:

examples/main_lllist.c.

4.3.4.5 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.

Examples:

examples/main_lllist.c.

4.3.4.6 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. 52)
_gdsl_list_remove() (p. 53)

4.3.4.7 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. 52)
_gdsl_list_remove() (p. 53)

4.3.4.8 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. 52)
_gdsl_list_insert_before() (p. 52)

4.3.4.9 _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.

4.3.4.10 _gdsl_list_t _gdsl_list_map_forward (const _gdsl_list_t *L*, const _gdsl_node_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. 54) 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_DAT-A	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. 55)

Examples:

examples/main_lllist.c.

4.3.4.11 _gdsl_list_t _gdsl_list_map_backward (const _gdsl_list_t *L*, const _gdsl_node_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. 55) 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. 54)

Examples:

examples/main_lllist.c.

4.3.4.12 void _gdsl_list_write (const _gdsl_list_t *L*, const _gdsl_node_write_func_t *WRITE_F*, FILE ∗ *OUTPUT_FILE*, void ∗ *USER_DATA*)

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

Write the nodes 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_F-ILE	The file where to write L's nodes.
USER_DAT-A	User's datas passed to WRITE_F.

See also

_gdsl_list_write_xml() (p. 56)
_gdsl_list_dump() (p. 57)

Examples:

examples/main_lllist.c.

4.3.4.13 void _gdsl_list_write_xml (const _gdsl_list_t *L*, const _gdsl_node_write_func_t *WRITE_F*, FILE ∗ *OUTPUT_FILE*, void ∗ *USER_DATA*)

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

Write the nodes 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 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. 55)
_gdsl_list_dump() (p. 57)

Examples:

examples/main_lllist.c.

4.3.4.14 void _gdsl_list_dump (const _gdsl_list_t *L*, const _gdsl_node_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 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. 55)
_gdsl_list_write_xml() (p. 56)

Examples:

examples/main_lllist.c.

4.4 Low-level doubly-linked node manipulation module.

This module is for manipulation of low-level doubly-linked nodes.

Typedefs

- typedef struct _gdsl_node ∗ **_gdsl_node_t**

  GDSL low-level doubly linked node type.
- typedef int(∗ **_gdsl_node_map_func_t**)(const **_gdsl_node_t** NODE, void ∗USER_DATA)

  GDSL low-level doubly-linked node map function type.
- typedef void(∗ **_gdsl_node_write_func_t**)(const **_gdsl_node_t** NODE, FILE ∗OUTPUT_FILE, void ∗USER_DATA)

  GDSL low-level doubly-linked node write function 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_node_write_func_t** WRITE_F, FILE ∗OUTPUT_FILE, void ∗USER_DATA)

  Write a low-level node to a file.

- void **_gdsl_node_write_xml** (const **_gdsl_node_t** NODE, const **_gdsl_node_-write_func_t** WRITE_F, FILE ∗OUTPUT_FILE, void ∗USER_DATA)

  Write a low-level node to a file into XML.
- void **_gdsl_node_dump** (const **_gdsl_node_t** NODE, const **_gdsl_node_write-_func_t** WRITE_F, FILE ∗OUTPUT_FILE, void ∗USER_DATA)

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

4.4.1 Detailed Description

This module is for manipulation of low-level doubly-linked nodes.

4.4.2 Copyright

4.4.3 Typedef Documentation

4.4.3.1 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 56 of file _gdsl_node.h.

4.4.3.2 typedef int(∗ _gdsl_node_map_func_t)(const _gdsl_node_t NODE, void ∗USER_DATA)

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

Parameters

NODE	The low-level node to map.
USER_DAT-A	The user datas to pass to this function.

Returns

GDSL_MAP_STOP if the mapping must be stopped.
GDSL_MAP_CONT if the mapping must be continued.

Definition at line 65 of file _gdsl_node.h.

4.4.3.3 typedef void(∗ _gdsl_node_write_func_t)(const _gdsl_node_t NODE, FILE ∗OUTPUT_FILE, void ∗USER_DATA)

GDSL low-level doubly-linked node write function type.

Parameters

TREE	The low-level doubly-linked node to write.
OUTPUT_FILE	The file where to write NODE.
USER_DATA	The user datas to pass to this function.

Definition at line 75 of file _gdsl_node.h.

4.4.4 Function Documentation

4.4.4.1 _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. 62)

4.4.4.2 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. 61)

4.4.4.3 _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. 63)
_gdsl_node_set_succ() (p. 64)
_gdsl_node_set_pred() (p. 64)

4.4.4.4 _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. 62)
_gdsl_node_set_succ() (p. 64)
_gdsl_node_set_pred() (p. 64)

4.4.4.5 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. 65)

Examples:

examples/main_lllist.c.

4.4.4.6 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. 62)

4.4.4.7 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. 63)

4.4.4.8 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. 63)

4.4.4.9 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. 66)

4.4.4.10 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. 65)

4.4.4.11 void _gdsl_node_write (const _gdsl_node_t *NODE*, const _gdsl_node_write_func_t *WRITE_F*, FILE ∗ *OUTPUT_FILE*, void ∗ *USER_DATA*)

Write a low-level node to a file.

Write 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.
USER_DATA	User's datas passed to WRITE_F.

See also

_gdsl_node_write_xml() (p. 67)
_gdsl_node_dump() (p. 68)

4.4.4.12 void _gdsl_node_write_xml (const _gdsl_node_t *NODE*, const _gdsl_node_write_func_t *WRITE_F*, FILE * *OUTPUT_FILE*, void * *USER_DATA*)

Write a low-level node to a file into XML.

Write the low-level node NODE to OUTPUT_FILE, into XML language. If WRITE_F != NULL, then uses WRITE_F function to write NODE 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.
USER_DATA	User's datas passed to WRITE_F.

See also

_gdsl_node_write() (p. 66)
_gdsl_node_dump() (p. 68)

4.4.4.13 void _gdsl_node_dump (const _gdsl_node_t *NODE*, const _gdsl_node_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 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.
USER_DATA	User's datas passed to WRITE_F.

See also

_gdsl_node_write() (p. 66)
_gdsl_node_write_xml() (p. 67)

4.5 Main module

GDSL main module.

Functions

- const char * **gdsl_get_version** (void)

  Get GDSL version number as a string.

4.5.1 Detailed Description

GDSL main module.

4.5.2 Copyright

4.5.3 Function Documentation

4.5.3.1 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.

4.6 2D-Arrays manipulation module.

This module is for manipulation of 2D-arrays.

Typedefs

- typedef struct gdsl_2darray ∗ **gdsl_2darray_t**

  GDSL 2D-array type.

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 ∗NE-W_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_write_func_t** W-RITE_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.

4.6.1 Detailed Description

This module is for manipulation of 2D-arrays.

4.6.2 Copyright

4.6.3 Typedef Documentation

4.6.3.1 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 56 of file gdsl_2darray.h.

4.6.4 Function Documentation

4.6.4.1 **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. 73)
gdsl_alloc_func_t (p. 247)
gdsl_free_func_t (p. 248)

Examples:

examples/main_2darray.c.

4.6.4.2 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. 72) 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. 72)

Examples:

examples/main_2darray.c.

4.6.4.3 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. 77)

Examples:

examples/main_2darray.c.

4.6.4.4 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. 75)
gdsl_2darray_get_size() (p. 76)

Examples:

examples/main_2darray.c.

4.6.4.5 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. 74)
gdsl_2darray_get_size() (p. 76)

Examples:

examples/main_2darray.c.

4.6.4.6 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. 74)
gdsl_2darray_get_columns_number() (p. 75)

4.6.4.7 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. 77)

4.6.4.8 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. 74)

4.6.4.9 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. 76)

Examples:

examples/main_2darray.c.

4.6.4.10 void gdsl_2darray_write (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.

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. 79)
gdsl_2darray_dump() (p. 80)

Examples:

examples/main_2darray.c.

4.6.4.11 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. 78)
gdsl_2darray_dump() (p. 80)

Examples:

examples/main_2darray.c.

4.6.4.12 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. 78)
gdsl_2darray_write_xml() (p. 79)

Examples:

examples/main_2darray.c.

4.7 Binary search tree manipulation module.

This module is for manipulation of binary search trees.

Typedefs

- typedef struct gdsl_bstree ∗ **gdsl_bstree_t**

  GDSL binary search tree type.

Functions

- **gdsl_bstree_t gdsl_bstree_alloc** (const char ∗NAME, **gdsl_alloc_func_t** ALL-OC_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_N-AME)

  Set the name of a binary search tree.
- **gdsl_element_t gdsl_bstree_insert** (**gdsl_bstree_t** T, void ∗VALUE, int ∗RES-ULT)

  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** WRIT-E_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.

4.7.1 Detailed Description

This module is for manipulation of binary search trees.

4.7.2 Copyright

This file is part of the Generic Data Structures Library (GDSL). Copyright (C) 1998-2018 Nicolas Darnis <ndarnis@free.fr>.

GDSL is free software: you can redistribute it and/or modify it under the terms of the GN-U General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

GDSL is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTI-CULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with GDSL. If not, see <http://www.gnu.org/licenses/>.

4.7.3 Typedef Documentation

4.7.3.1 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 56 of file gdsl_bstree.h.

4.7.4 Function Documentation

4.7.4.1 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. 84)
gdsl_bstree_flush() (p. 84)
gdsl_alloc_func_t (p. 247)
gdsl_free_func_t (p. 248)
gdsl_compare_func_t (p. 249)

Examples:

examples/main_bstree.c.

4.7.4.2 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. 83). 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. 83)
gdsl_bstree_flush() (p. 84)

Examples:

examples/main_bstree.c.

4.7.4.3 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. 214). 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. 83)
gdsl_bstree_free() (p. 84)

Examples:

examples/main_bstree.c.

4.7.4.4 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. 88) ()

4.7.4.5 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.

Examples:

examples/main_bstree.c.

4.7.4.6 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.

Examples:

examples/main_bstree.c.

4.7.4.7 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. 87)

Examples:

examples/main_bstree.c.

4.7.4.8 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. 86)

Examples:

examples/main_bstree.c.

4.7.4.9 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. 85)

4.7.4.10 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. 89)
gdsl_bstree_delete() (p. 90)

Examples:

examples/main_bstree.c.

4.7.4.11 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. 83). 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. 88)
gdsl_bstree_delete() (p. 90)

4.7.4.12 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. 83). 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. 83), 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. 88)
gdsl_bstree_remove() (p. 89)

Examples:

examples/main_bstree.c.

4.7.4.13 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. 83) 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. 88)
gdsl_bstree_remove() (p. 89)
gdsl_bstree_delete() (p. 90)

Examples:

examples/main_bstree.c.

4.7.4.14 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. 91) 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_DAT-A	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. 92)
gdsl_bstree_map_postfix() (p. 93)

4.7.4.15 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. 92) 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. 91)
gdsl_bstree_map_postfix() (p. 93)

4.7.4.16 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. 93) 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. 91)
gdsl_bstree_map_infix() (p. 92)

4.7.4.17 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. 94)
gdsl_bstree_dump() (p. 95)

Examples:

examples/main_bstree.c.

4.7.4.18 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. 94)
gdsl_bstree_dump() (p. 95)

Examples:

examples/main_bstree.c.

4.7.4.19 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. 94)
gdsl_bstree_write_xml() (p. 94)

Examples:

examples/main_bstree.c.

4.8 Hashtable manipulation module.

This module is for manipulation of hashtables.

Typedefs

- typedef struct hash_table * **gdsl_hash_t**

  GDSL hashtable type.
- typedef const char *(* **gdsl_key_func_t**)(void *VALUE)

  GDSL hashtable key function type.
- typedef **ulong**(* **gdsl_hash_func_t**)(const char *KEY)

  GDSL hashtable hash function type.

Functions

- **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** ALLO-C_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_NAM-E)

  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** M-AP_F, void ∗USER_DATA)

  Parse a hashtable.
- void **gdsl_hash_write** (const **gdsl_hash_t** H, **gdsl_write_func_t** WRITE_F, FI-LE ∗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, F-ILE ∗OUTPUT_FILE, void ∗USER_DATA)

  Dump the internal structure of a hashtable to a file.

4.8.1 Detailed Description

This module is for manipulation of hashtables.

4.8.2 Copyright

4.8.3 Typedef Documentation

4.8.3.1 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 56 of file gdsl_hash.h.

4.8.3.2 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 64 of file gdsl_hash.h.

4.8.3.3 typedef 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 72 of file gdsl_hash.h.

4.8.4 Function Documentation

4.8.4.1 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.

4.8.4.2 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. 109) 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. 99) 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. 101)
gdsl_hash_flush() (p. 101)
gdsl_hash_insert() (p. 106)
gdsl_hash_modify() (p. 109)

Examples:

examples/main_hash.c.

4.8.4.3 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. 100). 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. 100)
gdsl_hash_flush() (p. 101)

Examples:

examples/main_hash.c.

4.8.4.4 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. 100). 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. 100)
gdsl_hash_free() (p. 101)

Examples:

examples/main_hash.c.

4.8.4.5 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. 106)

4.8.4.6 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. 104)
gdsl_hash_fill_factor()

4.8.4.7 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. 109) 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. 103)
gdsl_hash_get_longest_list_size() (p. 104)
gdsl_hash_modify() (p. 109)

4.8.4.8 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. 104)
gdsl_hash_fill_factor()
gdsl_hash_get_entries_number() (p. 103)
gdsl_hash_get_lists_max_size() (p. 103)

4.8.4.9 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. 103)
gdsl_hash_fill_factor()
gdsl_hash_get_longest_list_size() (p. 104)

4.8.4.10 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. 103)
gdsl_hash_get_longest_list_size() (p. 104)
gdsl_hash_get_size() (p. 104)

Examples:

examples/main_hash.c.

4.8.4.11 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. 102)

4.8.4.12 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. 100).

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. 100)
gdsl_hash_remove() (p. 107)
gdsl_hash_delete() (p. 108)
gdsl_hash_get_size() (p. 104)
gdsl_hash_get_entries_number() (p. 103)
gdsl_hash_modify() (p. 109)

Examples:

examples/main_hash.c.

4.8.4.13 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. 106)
gdsl_hash_search() (p. 110)
gdsl_hash_delete() (p. 108)

Examples:

examples/main_hash.c.

4.8.4.14 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. 100), 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. 106)
gdsl_hash_search() (p. 110)
gdsl_hash_remove() (p. 107)

4.8.4.15 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. 106).

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. 106)
gdsl_hash_get_entries_number() (p. 103)
gdsl_hash_get_fill_factor() (p. 105)
gdsl_hash_get_longest_list_size() (p. 104)
gdsl_hash_get_lists_max_size() (p. 103)

4.8.4.16 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. 106)
gdsl_hash_remove() (p. 107)
gdsl_hash_delete() (p. 108)

Examples:

examples/main_hash.c.

4.8.4.17 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. 110) 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.

4.8.4.18 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. 112)
gdsl_hash_dump() (p. 112)

Examples:

examples/main_hash.c.

4.8.4.19 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. 111)
gdsl_hash_dump() (p. 112)

Examples:

examples/main_hash.c.

4.8.4.20 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. 111)
gdsl_hash_write_xml() (p. 112)

Examples:

examples/main_hash.c.

4.9 Heap manipulation module.

This module is for manipulation of heaps.

Typedefs

- typedef struct heap ∗ **gdsl_heap_t**

  GDSL heap type.

Functions

- **gdsl_heap_t gdsl_heap_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 heap.
- void **gdsl_heap_free** (**gdsl_heap_t** H)

  Destroy a heap.
- void **gdsl_heap_flush** (**gdsl_heap_t** H)

  Flush a heap.
- const char ∗ **gdsl_heap_get_name** (const **gdsl_heap_t** H)

  Get the name of a heap.
- **ulong gdsl_heap_get_size** (const **gdsl_heap_t** H)

  Get the size of a heap.
- **gdsl_element_t gdsl_heap_get_top** (const **gdsl_heap_t** H)

  Get the top of a heap.
- **bool gdsl_heap_is_empty** (const **gdsl_heap_t** H)

  Check if a heap is empty.
- **gdsl_heap_t gdsl_heap_set_name** (**gdsl_heap_t** H, const char ∗NEW_NAM-E)

  Set the name of a heap.
- **gdsl_element_t gdsl_heap_set_top** (**gdsl_heap_t** H, void ∗VALUE)

  Substitute the top element of a heap by a lesser one.
- **gdsl_element_t gdsl_heap_insert** (**gdsl_heap_t** H, void ∗VALUE)

  Insert an element into a heap (PUSH).
- **gdsl_element_t gdsl_heap_remove_top** (**gdsl_heap_t** H)

  Remove the top element from a heap (POP).
- **gdsl_heap_t gdsl_heap_delete_top** (**gdsl_heap_t** H)

  Delete the top element from a heap.
- **gdsl_element_t gdsl_heap_map_forward** (const **gdsl_heap_t** H, **gdsl_map_-func_t** MAP_F, void ∗USER_DATA)

  Parse a heap.
- void **gdsl_heap_write** (const **gdsl_heap_t** H, **gdsl_write_func_t** WRITE_F, FI-LE ∗OUTPUT_FILE, void ∗USER_DATA)

  Write all the elements of a heap to a file.

- void **gdsl_heap_write_xml** (const **gdsl_heap_t** H, **gdsl_write_func_t** WRITE-_F, FILE ∗OUTPUT_FILE, void ∗USER_DATA)

  Write the content of a heap to a file into XML.
- void **gdsl_heap_dump** (const **gdsl_heap_t** H, **gdsl_write_func_t** WRITE_F, F-ILE ∗OUTPUT_FILE, void ∗USER_DATA)

  Dump the internal structure of a heap to a file.

4.9.1 Detailed Description

This module is for manipulation of heaps.

4.9.2 Copyright

This file is part of the Generic Data Structures Library (GDSL). Copyright (C) 1998-2018 Nicolas Darnis <ndarnis@free.fr>.

GDSL is free software: you can redistribute it and/or modify it under the terms of the GN-U General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

GDSL is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTI-CULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with GDSL. If not, see <http://www.gnu.org/licenses/>.

4.9.3 Typedef Documentation

4.9.3.1 typedef struct heap∗ gdsl_heap_t

GDSL heap 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 57 of file gdsl_heap.h.

4.9.4 Function Documentation

4.9.4.1 gdsl_heap_t gdsl_heap_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 heap.

Allocate a new heap 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 heap. 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 heap to create
ALLOC_F	Function to alloc element when inserting it in the heap
FREE_F	Function to free element when removing it from the heap
COMP_F	Function to compare elements into the heap

Returns

the newly allocated heap in case of success.
NULL in case of insufficient memory.

See also

gdsl_heap_free() (p. 116)
gdsl_heap_flush() (p. 117)

Examples:

examples/main_heap.c.

4.9.4.2 void gdsl_heap_free (gdsl_heap_t *H*)

Destroy a heap.

Deallocate all the elements of the heap H by calling H's FREE_F function passed to **gdsl_heap_alloc()** (p. 115). The name of H is deallocated and H is deallocated itself too.

Note

Complexity: O(|H|)

Precondition

H must be a valid gdsl_heap_t

Parameters

H	The heap to destroy

See also

gdsl_heap_alloc() (p. 115)
gdsl_heap_flush() (p. 117)

Examples:

examples/main_heap.c.

4.9.4.3 void gdsl_heap_flush (gdsl_heap_t *H*)

Flush a heap.

Deallocate all the elements of the heap H by calling H's FREE_F function passed to **gdsl_heap_alloc()** (p. 115). H is not deallocated itself and H's name is not modified.

Note

Complexity: O(|H|)

Precondition

H must be a valid gdsl_heap_t

Parameters

H	The heap to flush

See also

gdsl_heap_alloc() (p. 115)
gdsl_heap_free() (p. 116)

Examples:

examples/main_heap.c.

4.9.4.4 const char∗ gdsl_heap_get_name (const gdsl_heap_t *H*)

Get the name of a heap.

Note

Complexity: O(1)

Precondition

H must be a valid gdsl_heap_t

Postcondition

The returned string MUST NOT be freed.

Parameters

H	The heap to get the name from

Returns

the name of the heap H.

See also

gdsl_heap_set_name() (p. 120)

Examples:

examples/main_heap.c.

4.9.4.5 ulong gdsl_heap_get_size (const gdsl_heap_t *H*)

Get the size of a heap.

Note

Complexity: O(1)

Precondition

H must be a valid gdsl_heap_t

Parameters

H	The heap to get the size from

Returns

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

4.9.4.6 gdsl_element_t gdsl_heap_get_top (const gdsl_heap_t *H*)

Get the top of a heap.

Note

Complexity: O(1)

Precondition

H must be a valid gdsl_heap_t

Parameters

H	The heap to get the top from

Returns

the element contained at the top position of the heap H if H is not empty. The returned element is not removed from H.
NULL if the heap H is empty.

See also

gdsl_heap_set_top() (p. 120)

Examples:

examples/main_heap.c.

4.9.4.7 bool gdsl_heap_is_empty (const gdsl_heap_t *H*)

Check if a heap is empty.

Note

Complexity: O(1)

Precondition

H must be a valid gdsl_heap_t

Parameters

H	The heap to check

Returns

TRUE if the heap H is empty.
FALSE if the heap H is not empty.

Examples:

examples/main_heap.c.

4.9.4.8 gdsl_heap_t gdsl_heap_set_name (gdsl_heap_t *H*, const char ∗ *NEW_NAME*)

Set the name of a heap.

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

Note

Complexity: O(1)

Precondition

H must be a valid gdsl_heap_t

Parameters

H	The heap to change the name
NEW_NAME	The new name of H

Returns

the modified heap in case of success.
NULL in case of insufficient memory.

See also

gdsl_heap_get_name() (p. 117)

4.9.4.9 gdsl_element_t gdsl_heap_set_top (gdsl_heap_t *H*, void ∗ *VALUE*)

Substitute the top element of a heap by a lesser one.

Try to replace the top element of a heap by a lesser one.

Note

Complexity: O(log (|H|))

Precondition

H must be a valid gdsl_heap_t

Parameters

H	The heap to substitute the top element
VALUE	the value to substitute to the top

Returns

The old top element value in case VALUE is lesser than all other H elements. NULL in case of VALUE is greather or equal to all other H elements.

See also

gdsl_heap_get_top() (p. 119)

Examples:

examples/main_heap.c.

4.9.4.10 gdsl_element_t gdsl_heap_insert (gdsl_heap_t *H*, void ∗ *VALUE*)

Insert an element into a heap (PUSH).

Allocate a new element E by calling H's ALLOC_F function on VALUE. The element E is then inserted into H at the good position to ensure H is always a heap.

Note

Complexity: O(log (|H|))

Precondition

H must be a valid gdsl_heap_t

Parameters

H	The heap 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_heap_alloc() (p. 115)
gdsl_heap_remove()
gdsl_heap_delete()
gdsl_heap_get_size() (p. 118)

Examples:

examples/main_heap.c.

4.9.4.11 gdsl_element_t gdsl_heap_remove_top (gdsl_heap_t *H*)

Remove the top element from a heap (POP).

Remove the top element from the heap H. The element is removed from H and is also returned.

Note

Complexity: O(log (|H|))

Precondition

H must be a valid gdsl_heap_t

Parameters

H	The heap to modify

Returns

the removed top element.
NULL if the heap is empty.

See also

gdsl_heap_insert() (p. 121)
gdsl_heap_delete_top() (p. 122)

4.9.4.12 gdsl_heap_t gdsl_heap_delete_top (gdsl_heap_t *H*)

Delete the top element from a heap.

Remove the top element from the heap H. The element is removed from H and is also deallocated using H's FREE_F function passed to **gdsl_heap_alloc()** (p. 115), then H is returned.

Note

Complexity: O(log (|H|))

Precondition

H must be a valid gdsl_heap_t

Parameters

H	The heap to modify

Returns

the modified heap after removal of top element.
NULL if heap is empty.

See also

gdsl_heap_insert() (p. 121)
gdsl_heap_remove_top() (p. 122)

Examples:

examples/main_heap.c.

4.9.4.13 gdsl_element_t gdsl_heap_map_forward (const gdsl_heap_t *H*, gdsl_map_func_t *MAP_F*, void * *USER_DATA*)

Parse a heap.

Parse all elements of the heap 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_heap_-map() stops and returns its last examinated element.

Note

Complexity: O(|H|)

Precondition

H must be a valid gdsl_heap_t & MAP_F != NULL

Parameters

H	The heap 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.

Examples:

examples/main_heap.c.

4.9.4.14 void gdsl_heap_write (const gdsl_heap_t *H*, gdsl_write_func_t *WRITE_F*, FILE ∗ *OUTPUT_FILE*, void ∗ *USER_DATA*)

Write all the elements of a heap to a file.

Write the elements of the heap 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_heap_t & OUTPUT_FILE != NULL & WRITE_F != NULL

Parameters

H	The heap 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_heap_write_xml() (p. 125)
gdsl_heap_dump() (p. 125)

4.9.4.15 void gdsl_heap_write_xml (const gdsl_heap_t *H*, gdsl_write_func_t *WRITE_F*, FILE ∗ *OUTPUT_FILE*, void ∗ *USER_DATA*)

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

Write the elements of the heap 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_heap_t & OUTPUT_FILE != NULL

Parameters

H	The heap to write.
WRITE_F	The write function.
OUTPUT_F-ILE	The file where to write H's elements.
USER_DAT-A	User's datas passed to WRITE_F.

See also

gdsl_heap_write() (p. 124)
gdsl_heap_dump() (p. 125)

Examples:

examples/main_heap.c.

4.9.4.16 void gdsl_heap_dump (const gdsl_heap_t *H*, gdsl_write_func_t *WRITE_F*, FILE ∗ *OUTPUT_FILE*, void ∗ *USER_DATA*)

Dump the internal structure of a heap to a file.

Dump the structure of the heap 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_heap_t & OUTPUT_FILE != NULL

Parameters

H	The heap to write
WRITE_F	The write function
OUTPUT_F-ILE	The file where to write H's elements
USER_DAT-A	User's datas passed to WRITE_F

See also

gdsl_heap_write() (p. 124)
gdsl_heap_write_xml() (p. 125)

Examples:

examples/main_heap.c.

4.10 Interval Heap manipulation module.

This module is for manipulation of interval heaps.

Typedefs

- typedef struct heap ∗ **gdsl_interval_heap_t**

  GDSL interval heap type.

Functions

- **gdsl_interval_heap_t gdsl_interval_heap_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 interval heap.
- void **gdsl_interval_heap_free** (**gdsl_interval_heap_t** H)

  Destroy an interval heap.
- void **gdsl_interval_heap_flush** (**gdsl_interval_heap_t** H)

  Flush an interval heap.
- const char ∗ **gdsl_interval_heap_get_name** (const **gdsl_interval_heap_t** H)

  Get the name of an interval heap.
- **ulong gdsl_interval_heap_get_size** (const **gdsl_interval_heap_t** H)

  Get the size of a interval heap.
- void **gdsl_interval_heap_set_max_size** (const **gdsl_interval_heap_t** H, **ulong** size)

  Set the maximum size of the interval heap.
- **bool gdsl_interval_heap_is_empty** (const **gdsl_interval_heap_t** H)

  Check if an interval heap is empty.
- **gdsl_interval_heap_t gdsl_interval_heap_set_name** (**gdsl_interval_heap_t** H, const char ∗NEW_NAME)

  Set the name of an interval heap.
- **gdsl_element_t gdsl_interval_heap_insert** (**gdsl_interval_heap_t** H, void ∗VALUE)

  Insert an element into an interval heap (PUSH).
- **gdsl_element_t gdsl_interval_heap_remove_max** (**gdsl_interval_heap_t** H)

  Remove the maximum element from an interval heap (POP).
- **gdsl_element_t gdsl_interval_heap_remove_min** (**gdsl_interval_heap_t** H)

  Remove the minimum element from an interval heap (POP).
- **gdsl_element_t gdsl_interval_heap_get_min** (const **gdsl_interval_heap_t** H)

  Get the minimum element.

- **gdsl_element_t gdsl_interval_heap_get_max** (const **gdsl_interval_heap_t** H)

  Get the maximum element.
- **gdsl_interval_heap_t gdsl_interval_heap_delete_min** (**gdsl_interval_heap_t** H)

  Delete the minimum element from an interval heap.
- **gdsl_interval_heap_t gdsl_interval_heap_delete_max** (**gdsl_interval_heap_t** H)

  Delete the maximum element from an interval heap.
- **gdsl_element_t gdsl_interval_heap_map_forward** (const **gdsl_interval_heap_t** H, **gdsl_map_func_t** MAP_F, void ∗USER_DATA)

  Parse a interval heap.
- void **gdsl_interval_heap_write** (const **gdsl_interval_heap_t** H, **gdsl_write_func_t** WRITE_F, FILE ∗OUTPUT_FILE, void ∗USER_DATA)

  Write all the elements of an interval heap to a file.
- void **gdsl_interval_heap_write_xml** (const **gdsl_interval_heap_t** H, **gdsl_write_func_t** WRITE_F, FILE ∗OUTPUT_FILE, void ∗USER_DATA)

  Write the content of an interval heap to a file into XML.
- void **gdsl_interval_heap_dump** (const **gdsl_interval_heap_t** H, **gdsl_write_func_t** WRITE_F, FILE ∗OUTPUT_FILE, void ∗USER_DATA)

  Dump the internal structure of an interval heap to a file.

4.10.1 Detailed Description

This module is for manipulation of interval heaps.

4.10.2 Copyright

4.10.3 Typedef Documentation

4.10.3.1 typedef struct heap∗ gdsl_interval_heap_t

GDSL interval heap type.

This type is voluntary opaque. Variables of this kind couldn't be directly used, but by the functions of this module.

Definition at line 56 of file gdsl_interval_heap.h.

4.10.4 Function Documentation

4.10.4.1 gdsl_interval_heap_t gdsl_interval_heap_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 interval heap.

Allocate a new interval heap 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 interval heap. 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 interval heap to create
ALLOC_F	Function to alloc element when inserting it in the interval heap
FREE_F	Function to free element when removing it from the interval heap
COMP_F	Function to compare elements into the interval heap

Returns

the newly allocated interval heap in case of success.
NULL in case of insufficient memory.

See also

gdsl_interval_heap_free() (p. 130)
gdsl_interval_heap_flush() (p. 130)

Examples:

examples/main_interval_heap.c.

4.10.4.2 void gdsl_interval_heap_free (gdsl_interval_heap_t *H*)

Destroy an interval heap.

Deallocate all the elements of the interval heap H by calling H's FREE_F function passed to **gdsl_interval_heap_alloc()** (p. 129). The name of H is deallocated and - H is deallocated itself too.

Note

Complexity: O(|H|)

Precondition

H must be a valid gdsl_interval_heap_t

Parameters

H	The interval heap to destroy

See also

gdsl_interval_heap_alloc() (p. 129)
gdsl_interval_heap_flush() (p. 130)

Examples:

examples/main_interval_heap.c.

4.10.4.3 void gdsl_interval_heap_flush (gdsl_interval_heap_t *H*)

Flush an interval heap.

Deallocate all the elements of the interval heap H by calling H's FREE_F function passed to **gdsl_interval_heap_alloc()** (p. 129). H is not deallocated itself and H's name is not modified.

Note

Complexity: O(|H|)

Precondition

H must be a valid gdsl_interval_heap_t

Parameters

H	The heap to flush

See also

gdsl_interval_heap_alloc() (p. 129)
gdsl_interval_heap_free() (p. 130)

Examples:

examples/main_interval_heap.c.

4.10.4.4 const char∗ gdsl_interval_heap_get_name (const gdsl_interval_heap_t *H*)

Get the name of an interval heap.

Note

Complexity: O(1)

Precondition

H must be a valid gdsl_interval_heap_t

Postcondition

The returned string MUST NOT be freed.

Parameters

H	The interval heap to get the name from

Returns

the name of the interval heap H.

See also

gdsl_interval_heap_set_name() (p. 133)

4.10.4.5 ulong gdsl_interval_heap_get_size (const gdsl_interval_heap_t *H*)

Get the size of a interval heap.

Note

Complexity: O(1)

Precondition

H must be a valid gdsl_interval_heap_t

Parameters

H	The interval heap to get the size from

Returns

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

Examples:

examples/main_interval_heap.c.

4.10.4.6 void gdsl_interval_heap_set_max_size (const gdsl_interval_heap_t *H*, ulong *size*)

Set the maximum size of the interval heap.

Note

Complexity: O(1)

Precondition

H must be a valid gdsl_interval_heap_t

Parameters

H	The interval heap to get the size from
size	The new maximum size

Returns

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

4.10.4.7 bool gdsl_interval_heap_is_empty (const gdsl_interval_heap_t *H*)

Check if an interval heap is empty.

Note

Complexity: O(1)

Precondition

H must be a valid gdsl_interval_heap_t

Parameters

H	The interval heap to check

Returns

TRUE if the interval heap H is empty.
FALSE if the interval heap H is not empty.

4.10.4.8 gdsl_interval_heap_t gdsl_interval_heap_set_name (gdsl_interval_heap_t *H*, const char ∗ *NEW_NAME*)

Set the name of an interval heap.

Change the previous name of the interval heap H to a copy of NEW_NAME.

Note

Complexity: O(1)

Precondition

H must be a valid gdsl_interval_heap_t

Parameters

H	The interval heap to change the name
NEW_NAME	The new name of H

Returns

the modified interval heap in case of success.
NULL in case of insufficient memory.

See also

gdsl_interval_heap_get_name() (p. 131)

4.10.4.9 gdsl_element_t gdsl_interval_heap_insert (gdsl_interval_heap_t *H*, void ∗ *VALUE*)

Insert an element into an interval heap (PUSH).

Allocate a new element E by calling H's ALLOC_F function on VALUE. The element E is then inserted into H at the good position to ensure H is always an interval heap.

Note

Complexity: O(log (|H|))

Precondition

H must be a valid gdsl_interval_heap_t

Parameters

H	The interval heap 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_interval_heap_alloc() (p. 129)
gdsl_interval_heap_remove()
gdsl_interval_heap_delete()
gdsl_interval_heap_get_size() (p. 131)

Examples:

examples/main_interval_heap.c.

4.10.4.10 gdsl_element_t gdsl_interval_heap_remove_max (gdsl_interval_heap_t *H*)

Remove the maximum element from an interval heap (POP).

Remove the maximum element from the interval heap H. The element is removed from H and is also returned.

Note

Complexity: O(log (|H|))

Precondition

H must be a valid gdsl_interval_heap_t

Parameters

H	The interval heap to modify

Returns

the removed top element.
NULL if the interval heap is empty.

See also

gdsl_interval_heap_insert() (p. 134)
gdsl_interval_heap_delete_max() (p. 137)

Examples:

examples/main_interval_heap.c.

4.10.4.11 gdsl_element_t gdsl_interval_heap_remove_min (gdsl_interval_heap_t *H*)

Remove the minimum element from an interval heap (POP).

Remove the minimum element from the interval heap H. The element is removed from H and is also returned.

Note

Complexity: O(log (|H|))

Precondition

H must be a valid gdsl_interval_heap_t

Parameters

H	The interval heap to modify

Returns

the removed top element.
NULL if the interval heap is empty.

See also

gdsl_interval_heap_insert() (p. 134)
gdsl_interval_heap_delete_max() (p. 137)

Examples:

examples/main_interval_heap.c.

4.10.4.12 gdsl_element_t gdsl_interval_heap_get_min (const gdsl_interval_heap_t *H*)

Get the minimum element.

Note

Complexity: O(1)

Precondition

H must be a valid gdsl_interval_heap_t

Parameters

H	The interval heap to get the size from

Returns

The smallest element in H

4.10.4.13 gdsl_element_t gdsl_interval_heap_get_max (const gdsl_interval_heap_t *H*)

Get the maximum element.

Note

Complexity: O(1)

Precondition

H must be a valid gdsl_interval_heap_t

Parameters

H	The interval heap to get the size from

Returns

The largest element in H

4.10.4.14 gdsl_interval_heap_t gdsl_interval_heap_delete_min (gdsl_interval_heap_t *H*)

Delete the minimum element from an interval heap.

Remove the minimum element from the interval heap H. The element is removed from H and is also deallocated using H's FREE_F function passed to **gdsl_interval_heap_-alloc()** (p. 129), then H is returned.

Note

Complexity: O(log ($|H|$))

Precondition

H must be a valid gdsl_interval_heap_t

Parameters

H	The interval heap to modify

Returns

the modified interval heap after removal of top element.
NULL if interval heap is empty.

See also

gdsl_interval_heap_insert() (p. 134)
gdsl_interval_heap_remove_top()

4.10.4.15 gdsl_interval_heap_t gdsl_interval_heap_delete_max (gdsl_interval_heap_t *H*)

Delete the maximum element from an interval heap.

Remove the maximum element from the interval heap H. The element is removed from H and is also deallocated using H's FREE_F function passed to **gdsl_interval_heap_-alloc()** (p. 129), then H is returned.

Note

Complexity: O(log ($|H|$))

Precondition

H must be a valid gdsl_interval_heap_t

Parameters

H	The interval heap to modify

Returns

the modified interval heap after removal of top element.
NULL if interval heap is empty.

See also

gdsl_interval_heap_insert() (p. 134)
gdsl_interval_heap_remove_top()

4.10.4.16 gdsl_element_t gdsl_interval_heap_map_forward (const gdsl_interval_heap_t *H*, gdsl_map_func_t *MAP_F*, void * *USER_DATA*)

Parse a interval heap.

Parse all elements of the interval heap 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_interval_heap_map() stops and returns its last examinated element.

Note

Complexity: O(|H|)

Precondition

H must be a valid gdsl_interval_heap_t & MAP_F != NULL

Parameters

H	The interval heap 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.

4.10.4.17 void gdsl_interval_heap_write (const gdsl_interval_heap_t *H*, gdsl_write_func_t *WRITE_F*, FILE ∗ *OUTPUT_FILE*, void ∗ *USER_DATA*)

Write all the elements of an interval heap to a file.

Write the elements of the interval heap 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_interval_heap_t & OUTPUT_FILE != NULL & WRITE_F != NULL

Parameters

H	The interval heap 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_interval_heap_write_xml() (p. 139)
gdsl_interval_heap_dump() (p. 140)

4.10.4.18 void gdsl_interval_heap_write_xml (const gdsl_interval_heap_t *H*, gdsl_write_func_t *WRITE_F*, FILE ∗ *OUTPUT_FILE*, void ∗ *USER_DATA*)

Write the content of an interval heap to a file into XML.

Write the elements of the interval heap 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_interval_heap_t & OUTPUT_FILE != NULL

Parameters

H	The interval heap 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_interval_heap_write() (p. 139)
gdsl_interval_heap_dump() (p. 140)

4.10.4.19 void gdsl_interval_heap_dump (const gdsl_interval_heap_t *H*, gdsl_write_func_t *WRITE_F*, FILE ∗ *OUTPUT_FILE*, void ∗ *USER_DATA*)

Dump the internal structure of an interval heap to a file.

Dump the structure of the interval heap 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_interval_heap_t & OUTPUT_FILE != NULL

Parameters

H	The interval heap 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_interval_heap_write() (p. 139)
gdsl_interval_heap_write_xml() (p. 139)

4.11 Doubly-linked list manipulation module.

This module is for manipulation of doubly-linked lists.

Typedefs

- typedef struct _gdsl_list * **gdsl_list_t**

  GDSL doubly-linked list type.
- typedef struct _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** P-OS)

  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)

  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, F-ILE ∗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 ∗V-ALUE)

    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.

4.11.1 Detailed Description

This module is for manipulation of doubly-linked lists.

4.11.2 Copyright

4.11.3 Typedef Documentation

4.11.3.1 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 54 of file gdsl_list.h.

4.11.3.2 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 62 of file gdsl_list.h.

4.11.4 Function Documentation

4.11.4.1 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. 145)
gdsl_list_flush() (p. 146)

Examples:

examples/main_list.c.

4.11.4.2 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. 144).

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. 144)
gdsl_list_flush() (p. 146)

Examples:

examples/main_list.c.

4.11.4.3 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. 144). 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. 144)
gdsl_list_free() (p. 145)

Examples:

examples/main_list.c.

4.11.4.4 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. 149)

Examples:

examples/main_list.c.

4.11.4.5 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|).

Examples:

examples/main_list.c.

4.11.4.6 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.

Examples:

examples/main_list.c.

4.11.4.7 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. 149)

4.11.4.8 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. 148)

4.11.4.9 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. 146)

4.11.4.10 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. 144). 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. 151)
gdsl_list_remove_head() (p. 151)
gdsl_list_remove_tail() (p. 152)
gdsl_list_remove() (p. 153)

Examples:

examples/main_list.c.

4.11.4.11 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. 144). 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. 150)
gdsl_list_remove_head() (p. 151)
gdsl_list_remove_tail() (p. 152)
gdsl_list_remove() (p. 153)

Examples:

examples/main_list.c.

4.11.4.12 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. 150)
gdsl_list_insert_tail() (p. 151)
gdsl_list_remove_tail() (p. 152)
gdsl_list_remove() (p. 153)

4.11.4.13 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. 150)
gdsl_list_insert_tail() (p. 151)
gdsl_list_remove_head() (p. 151)

gdsl_list_remove() (p. 153)

4.11.4.14 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. 150)
gdsl_list_insert_tail() (p. 151)
gdsl_list_remove_head() (p. 151)
gdsl_list_remove_tail() (p. 152)

4.11.4.15 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. 144).

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. 144)
gdsl_list_destroy_tail()
gdsl_list_destroy()

Examples:

examples/main_list.c.

4.11.4.16 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. 144).

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. 144)
gdsl_list_destroy_head()
gdsl_list_destroy()

Examples:

examples/main_list.c.

4.11.4.17 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. 144).

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. 144)
gdsl_list_destroy_head()
gdsl_list_destroy_tail()

Examples:

examples/main_list.c.

4.11.4.18 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. 156)
gdsl_list_search_max() (p. 157)
gdsl_list_search_min() (p. 158)

Examples:

examples/main_list.c.

4.11.4.19 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. 156)
gdsl_list_search_max() (p. 157)
gdsl_list_search_min() (p. 158)

Examples:

examples/main_list.c.

4.11.4.20 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. 156)
gdsl_list_search_by_position() (p. 156)
gdsl_list_search_min() (p. 158)

Examples:

examples/main_list.c.

4.11.4.21 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. 156)
gdsl_list_search_by_position() (p. 156)
gdsl_list_search_max() (p. 157)

4.11.4.22 gdsl_list_t gdsl_list_sort (gdsl_list_t *L*, gdsl_compare_func_t *COMP_F*)

Sort a list.

Sort the list L using COMP_F to order L's elements.

Note

Complexity: O(|L| * log(|L|))

Precondition

L must be a valid gdsl_list_t & COMP_F != NULL & L must not contains elements that are equals

Parameters

L	The list to sort
COMP_F	The comparison function used to order L's elements

Returns

the sorted list L.

Examples:

examples/main_list.c.

4.11.4.23 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. 159) 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. 160)

4.11.4.24 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. 160) stops and returns its last examined 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. 159)

Examples:

examples/main_list.c.

4.11.4.25 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. 161)
gdsl_list_dump() (p. 162)

4.11.4.26 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_F-ILE	The file where to write L's elements.
USER_DAT-A	User's datas passed to WRITE_F.

See also

gdsl_list_write() (p. 161)
gdsl_list_dump() (p. 162)

Examples:

examples/main_list.c.

4.11.4.27 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_F-ILE	The file where to write L's elements.
USER_DAT-A	User's datas passed to WRITE_F.

See also

gdsl_list_write() (p. 161)
gdsl_list_write_xml() (p. 161)

Examples:

examples/main_list.c.

4.11.4.28 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. 163)

Examples:

examples/main_list.c.

4.11.4.29 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. 163)

Examples:

examples/main_list.c.

4.11.4.30 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. 164)

Examples:

examples/main_list.c.

4.11.4.31 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. 164)

4.11.4.32 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. 166)

Examples:

examples/main_list.c.

4.11.4.33 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. 165)

4.11.4.34 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. 167)

Examples:

examples/main_list.c.

4.11.4.35 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. 166)

Examples:

examples/main_list.c.

4.11.4.36 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. 168)

4.11.4.37 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. 167)

4.11.4.38 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. 169)

4.11.4.39 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. 169)

4.11.4.40 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. 170) 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. 170)

4.11.4.41 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. 170)

Examples:

examples/main_list.c.

4.11.4.42 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. 144). 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. 172)
gdsl_list_cursor_remove_after() (p. 173)
gdsl_list_cursor_remove_before() (p. 174)

Examples:

examples/main_list.c.

4.11.4.43 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. 144). 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. 171)
gdsl_list_cursor_remove_after() (p. 173)
gdsl_list_cursor_remove_before() (p. 174)

Examples:

examples/main_list.c.

4.11.4.44 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. 171)
gdsl_list_cursor_insert_before() (p. 172)
gdsl_list_cursor_remove() (p. 172)
gdsl_list_cursor_remove_before() (p. 174)

4.11.4.45 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. 171)
gdsl_list_cursor_insert_before() (p. 172)
gdsl_list_cursor_remove() (p. 172)
gdsl_list_cursor_remove_before() (p. 174)

4.11.4.46 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. 171)
gdsl_list_cursor_insert_before() (p. 172)
gdsl_list_cursor_remove() (p. 172)
gdsl_list_cursor_remove_after() (p. 173)

4.11.4.47 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. 144).

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. 175)
gdsl_list_cursor_delete_after() (p. 175)

4.11.4.48 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. 144).

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. 174)
gdsl_list_cursor_delete_before() (p. 175)

Examples:

examples/main_list.c.

4.11.4.49 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. 144).

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. 174)
gdsl_list_cursor_delete_after() (p. 175)

4.12 Various macros module.

This module provides some various macros.

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.

4.12.1 Detailed Description

This module provides some various macros.

4.12.2 Copyright

4.12.3 Define Documentation

4.12.3.1 #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. 178)

Definition at line 59 of file gdsl_macros.h.

4.12.3.2 #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. 177)

Definition at line 76 of file gdsl_macros.h.

4.13 Permutation manipulation module.

This module is for manipulation of permutations.

Typedefs

- typedef struct gdsl_perm ∗ **gdsl_perm_t**

  GDSL permutation type.
- typedef void(∗ **gdsl_perm_write_func_t**)(**ulong** E, FILE ∗OUTPUT_FILE, **gdsl-_location_t** POSITION, void ∗USER_DATA)

  GDSL permutation write function type.
- typedef struct gdsl_perm_data ∗ **gdsl_perm_data_t**

Enumerations

- enum **gdsl_perm_position_t** { **GDSL_PERM_POSITION_FIRST** = 1, **GDSL_P-ERM_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_NAM-E)

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_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** W-RITE_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** WRIT-E_F, FILE ∗OUTPUT_FILE, void ∗USER_DATA)

  Dump the internal structure of a permutation to a file.

4.13.1 Detailed Description

This module is for manipulation of permutations.

4.13.2 Copyright

4.13.3 Typedef Documentation

4.13.3.1 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 53 of file gdsl_perm.h.

4.13.3.2 typedef void(∗ gdsl_perm_write_func_t)(ulong E, FILE ∗OUTPUT_FILE, gdsl_location_t POSITION, void ∗USER_DATA)

GDSL permutation write function type.

Parameters

E	The permutation element to write
OUTPUT_F-ILE	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_DAT-A	User's datas

Definition at line 77 of file gdsl_perm.h.

4.13.3.3 typedef struct gdsl_perm_data∗ gdsl_perm_data_t

Definition at line 83 of file gdsl_perm.h.

4.13.4 Enumeration Type Documentation

4.13.4.1 enum gdsl_perm_position_t

This type is for gdsl_perm_write_func_t.

Enumerator:

GDSL_PERM_POSITION_FIRST When element is at first position
GDSL_PERM_POSITION_LAST When element is at last position

Definition at line 58 of file gdsl_perm.h.

4.13.5 Function Documentation

4.13.5.1 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. 183)
gdsl_perm_copy() (p. 183)

Examples:

examples/main_bstree.c, **examples/main_list.c**, **examples/main_llbstree.c**, **examples/main_perm.c**, and **examples/main_rbtree.c**.

4.13.5.2 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. 182)
gdsl_perm_copy() (p. 183)

Examples:

examples/main_bstree.c, **examples/main_list.c**, **examples/main_llbstree.c**, **examples/main_perm.c**, and **examples/main_rbtree.c**.

4.13.5.3 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. 182)
gdsl_perm_free (p. 183)

4.13.5.4 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. 188)

4.13.5.5 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. 185)
gdsl_perm_get_elements_array() (p. 186)

4.13.5.6 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. 184)
gdsl_perm_get_elements_array() (p. 186)

Examples:

examples/main_bstree.c, **examples/main_list.c**, **examples/main_llbstree.c**, and **examples/main_rbtree.c**.

4.13.5.7 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. 185)
gdsl_perm_set_elements_array() (p. 190)

4.13.5.8 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.

Examples:

examples/main_perm.c.

4.13.5.9 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. 187)

Examples:

examples/main_perm.c.

4.13.5.10 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. 187)

Examples:

examples/main_perm.c.

4.13.5.11 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. 184)

4.13.5.12 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. 189)

4.13.5.13 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. 189)

4.13.5.14 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. 186)

4.13.5.15 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 RESU-LT 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 ALPHA x BETA.

4.13.5.16 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. 192)

Examples:

examples/main_perm.c.

4.13.5.17 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. 191)

Examples:

examples/main_perm.c.

4.13.5.18 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. 193)

Examples:

examples/main_perm.c.

4.13.5.19 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. 192)

Examples:

examples/main_perm.c.

4.13.5.20 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.

Examples:

examples/main_bstree.c, **examples/main_list.c**, **examples/main_llbstree.c**, **examples/main_perm.c**, and **examples/main_rbtree.c**.

4.13.5.21 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.

Examples:

examples/main_perm.c.

4.13.5.22 void gdsl_perm_write (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.

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. 195)
gdsl_perm_dump() (p. 196)

Examples:

examples/main_perm.c.

4.13.5.23 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 WR-ITE_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_F-ILE	The file where to write P's elements.
USER_DAT-A	User's datas passed to WRITE_F.

See also

gdsl_perm_write() (p. 195)
gdsl_perm_dump() (p. 196)

4.13.5.24 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_D-ATA 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_F-ILE	The file where to write P's elements.
USER_DAT-A	User's datas passed to WRITE_F.

See also

gdsl_perm_write() (p. 195)
gdsl_perm_write_xml() (p. 195)

4.14 Queue manipulation module.

This module is for manipulation of queues.

Typedefs

- typedef struct _gdsl_queue ∗ **gdsl_queue_t**

  GDSL queue type.

Functions

- **gdsl_queue_t gdsl_queue_alloc** (const char ∗NAME, **gdsl_alloc_func_t** ALL-OC_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_N-AME)

  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** WRI-TE_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.

4.14.1 Detailed Description

This module is for manipulation of queues.

4.14.2 Copyright

4.14.3 Typedef Documentation

4.14.3.1 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 57 of file gdsl_queue.h.

4.14.4 Function Documentation

4.14.4.1 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. 200)
gdsl_queue_flush() (p. 201)

Examples:

examples/main_queue.c.

4.14.4.2 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. 200). 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. 200)
gdsl_queue_flush() (p. 201)

Examples:

examples/main_queue.c.

4.14.4.3 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. 200)
gdsl_queue_free() (p. 200)

Examples:

examples/main_queue.c.

4.14.4.4 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. 204)

Examples:

examples/main_queue.c.

4.14.4.5 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|).

4.14.4.6 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.

Examples:

examples/main_queue.c.

4.14.4.7 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. 204)

Examples:

examples/main_queue.c.

4.14.4.8 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. 203)

Examples:

examples/main_queue.c.

4.14.4.9 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. 202)

4.14.4.10 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. 200). 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. 206)

Examples:

examples/main_queue.c.

4.14.4.11 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. 205)

Examples:

examples/main_queue.c.

4.14.4.12 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. 207)

4.14.4.13 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. 206)

Examples:

examples/main_queue.c.

4.14.4.14 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. 208) 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. 209)

Examples:

examples/main_queue.c.

4.14.4.15 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. 209) 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. 208)

4.14.4.16 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_F-ILE	The file where to write Q's elements.
USER_DAT-A	User's datas passed to WRITE_F.

See also

gdsl_queue_write_xml() (p. 210)
gdsl_queue_dump() (p. 211)

4.14.4.17 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_F-ILE	The file where to write Q's elements.
USER_DAT-A	User's datas passed to WRITE_F.

See also

gdsl_queue_write() (p. 209)
gdsl_queue_dump() (p. 211)

Examples:

examples/main_queue.c.

4.14.4.18 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. 209)
gdsl_queue_write_xml() (p. 210)

Examples:

examples/main_queue.c.

4.15 Red-black tree manipulation module.

This module is for manipulation of red-black trees.

Typedefs

- typedef struct 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** WRIT-E_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.

4.15.1 Detailed Description

This module is for manipulation of red-black trees.

4.15.2 Copyright

4.15.3 Typedef Documentation

4.15.3.1 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 55 of file gdsl_rbtree.h.

4.15.4 Function Documentation

4.15.4.1 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. 215)
gdsl_rbtree_flush() (p. 215)

Examples:

examples/main_rbtree.c.

4.15.4.2 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. 214). 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. 214)
gdsl_rbtree_flush() (p. 215)

Examples:

examples/main_rbtree.c.

4.15.4.3 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. 214). 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. 214)
gdsl_rbtree_free() (p. 215)

Examples:

examples/main_rbtree.c.

4.15.4.4 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. 219)

4.15.4.5 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.

Examples:

examples/main_rbtree.c.

4.15.4.6 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.

Examples:

examples/main_rbtree.c.

4.15.4.7 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()

Examples:

examples/main_rbtree.c.

4.15.4.8 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. 217)

Examples:

examples/main_rbtree.c.

4.15.4.9 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. 216)

4.15.4.10 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. 220)
gdsl_rbtree_delete() (p. 221)

Examples:

examples/main_rbtree.c.

4.15.4.11 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. 214). 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. 219)
gdsl_rbtree_delete() (p. 221)

4.15.4.12 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. 214). 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. 214), 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. 219)
gdsl_rbtree_remove() (p. 220)

Examples:

examples/main_rbtree.c.

4.15.4.13 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. 214) 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. 219)
gdsl_rbtree_remove() (p. 220)
gdsl_rbtree_delete() (p. 221)

Examples:

examples/main_rbtree.c.

4.15.4.14 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. 222) 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. 223)
gdsl_rbtree_map_postfix() (p. 224)

Examples:

examples/main_rbtree.c.

4.15.4.15 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. 223) 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_DAT-A	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. 222)
gdsl_rbtree_map_postfix() (p. 224)

Examples:

examples/main_rbtree.c.

4.15.4.16 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. 224) 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_DAT-A	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. 222)
gdsl_rbtree_map_infix() (p. 223)

Examples:

examples/main_rbtree.c.

4.15.4.17 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. 226)
gdsl_rbtree_dump() (p. 226)

Examples:

examples/main_rbtree.c.

4.15.4.18 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. 225)
gdsl_rbtree_dump() (p. 226)

Examples:

examples/main_rbtree.c.

4.15.4.19 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. 225)
gdsl_rbtree_write_xml() (p. 226)

Examples:

examples/main_rbtree.c.

4.16 Sort module.

This module is for sorting arrays.

Functions

- void **gdsl_sort** (**gdsl_element_t** ∗T, **ulong** N, const **gdsl_compare_func_t** COMP_F)
  Sort an array in place.

4.16.1 Detailed Description

This module is for sorting arrays.

4.16.2 Copyright

This file is part of the Generic Data Structures Library (GDSL). Copyright (C) 1998-2018 Nicolas Darnis <ndarnis@free.fr>.

GDSL is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

GDSL is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with GDSL. If not, see <http://www.gnu.org/licenses/>.

4.16.3 Function Documentation

4.16.3.1 void gdsl_sort (gdsl_element_t ∗ *T*, ulong *N*, const 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 & for all i <= N: sizeof (T[i]) == sizeof (gdsl_element_t)

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

Examples:

examples/main_sort.c.

4.17 Stack manipulation module.

This module is for manipulation of stacks.

Typedefs

- typedef struct _gdsl_stack ∗ **gdsl_stack_t**

  GDSL stack type.

Functions

- **gdsl_stack_t gdsl_stack_alloc** (const char ∗NAME, **gdsl_alloc_func_t** ALLO-C_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.
- **ulong 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_NAM-E)

  Set the name of a stack.
- void **gdsl_stack_set_growing_factor** (**gdsl_stack_t** S, **ulong** 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, F-ILE ∗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, FI-LE ∗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.

4.17.1 Detailed Description

This module is for manipulation of stacks.

4.17.2 Copyright

4.17.3 Typedef Documentation

4.17.3.1 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 56 of file gdsl_stack.h.

4.17.4 Function Documentation

4.17.4.1 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. 232)
gdsl_stack_flush() (p. 233)

Examples:

examples/main_stack.c.

4.17.4.2 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. 232). 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. 232)
gdsl_stack_flush() (p. 233)

Examples:

examples/main_stack.c.

4.17.4.3 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. 232). 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. 232)
gdsl_stack_free() (p. 232)

Examples:

examples/main_stack.c.

4.17.4.4 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. 237)

Examples:

examples/main_stack.c.

4.17.4.5 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|).

4.17.4.6 ulong 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. 238).

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. 238)
gdsl_stack_set_growing_factor() (p. 238)

4.17.4.7 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.

Examples:

examples/main_stack.c.

4.17.4.8 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. 237)

Examples:

examples/main_stack.c.

4.17.4.9 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. 236)

4.17.4.10 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. 234)

4.17.4.11 void gdsl_stack_set_growing_factor (gdsl_stack_t *S,* ulong *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. 235)

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. 238)
gdsl_stack_get_growing_factor() (p. 235)

4.17.4.12 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. 232). 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. 238)
gdsl_stack_get_growing_factor() (p. 235)
gdsl_stack_remove() (p. 239)

Examples:

examples/main_stack.c.

4.17.4.13 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. 238)

Examples:

examples/main_stack.c.

4.17.4.14 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. 241)

4.17.4.15 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. 240)

Examples:

examples/main_stack.c.

4.17.4.16 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_S-TOP, then **gdsl_stack_map_forward()** (p. 241) 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. 242)

Examples:

examples/main_stack.c.

4.17.4.17 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. 242) 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. 241)

4.17.4.18 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_F-ILE	The file where to write S's elements.
USER_DAT-A	User's datas passed to WRITE_F.

See also

gdsl_stack_write_xml() (p. 243)
gdsl_stack_dump() (p. 244)

4.17.4.19 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_F-ILE	The file where to write S's elements.
USER_DAT-A	User's datas passed to WRITE_F.

See also

gdsl_stack_write() (p. 243)
gdsl_stack_dump() (p. 244)

Examples:

examples/main_stack.c.

4.17.4.20 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_F-ILE	The file where to write S's elements.
USER_DAT-A	User's datas passed to WRITE_F.

See also

gdsl_stack_write() (p. 243)
gdsl_stack_write_xml() (p. 243)

Examples:

examples/main_stack.c.

4.18 GDSL types.

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, **gdsl_location_t** LOCATION, void ∗USER_DATA)

  GDSL Map element function type.
- typedef long 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, **gdsl_location_t** LOCATION, void ∗USER_DATA)

  GDSL Write element function type.
- typedef unsigned long int **ulong**
- typedef unsigned short int **ushort**

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 **gdsl_location_t** { **GDSL_LOCATION_UNDEF** = 0, **GDSL_LOCATION_HEAD** = 1, **GDSL_LOCATION_ROOT** = 1, **GDSL_LOCATION_TOP** = 1, × **GDSL_LOCATION_TAIL** = 2, **GDSL_LOCATION_LEAF** = 2, **GDSL_LOCATION_BOTTOM** = 2, **GDSL_LOCATION_FIRST** = 1, **GDSL_LOCATION_LAST** = 2, **GDSL_LOCATION_FIRST_COL** = 1, **GDSL_LOCATION_LAST_COL** = 2, **GDSL_LOCATION_FIRST_ROW** = 4, **GDSL_LOCATION_LAST_ROW** = 8 }
- enum **bool** { **FALSE** = 0, **TRUE** = 1 }

4.18.1 Detailed Description

GDSL types.

4.18.2 Copyright

4.18.3 Typedef Documentation

4.18.3.1 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 133 of file gdsl_types.h.

4.18.3.2 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. 248)

Definition at line 147 of file gdsl_types.h.

4.18.3.3 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. 247)

Definition at line 165 of file gdsl_types.h.

4.18.3.4 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 178 of file gdsl_types.h.

4.18.3.5 typedef int(∗ gdsl_map_func_t)(const gdsl_element_t E, gdsl_location_t LOCATION, 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.
LOCATION	The location of E in the data structure.
USER_DATA	User's datas.

Returns

GDSL_MAP_STOP if the mapping must be stopped.
GDSL_MAP_CONT if the mapping must be continued.

Definition at line 195 of file gdsl_types.h.

4.18.3.6 typedef long 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 216 of file gdsl_types.h.

4.18.3.7 typedef void(* gdsl_write_func_t)(const gdsl_element_t E, FILE *OUTPUT_FILE, gdsl_location_t LOCATION, 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 to write.
OUTPUT_FILE	The file where to write E.
LOCATION	The location of E in the data structure.
USER_DATA	User's datas.

Definition at line 232 of file gdsl_types.h.

4.18.3.8 typedef unsigned long int ulong

Definition at line 245 of file gdsl_types.h.

4.18.3.9 typedef unsigned short int ushort

Definition at line 249 of file gdsl_types.h.

4.18.4 Enumeration Type Documentation

4.18.4.1 enum gdsl_constant_t

GDSL Constants.

Enumerator:

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 51 of file gdsl_types.h.

4.18.4.2 enum gdsl_location_t

Enumerator:

GDSL_LOCATION_UNDEF Element position undefined

GDSL_LOCATION_HEAD Element is at head position

GDSL_LOCATION_ROOT Element is on leaf position

GDSL_LOCATION_TOP Element is at top position

GDSL_LOCATION_TAIL Element is at tail position

GDSL_LOCATION_LEAF Element is on root position

GDSL_LOCATION_BOTTOM Element is at bottom position

GDSL_LOCATION_FIRST Element is the first

GDSL_LOCATION_LAST Element is the last

GDSL_LOCATION_FIRST_COL Element is on first column

GDSL_LOCATION_LAST_COL Element is on last column

GDSL_LOCATION_FIRST_ROW Element is on first row

GDSL_LOCATION_LAST_ROW Element is on last row

Definition at line 72 of file gdsl_types.h.

4.18.4.3 enum bool

GDSL boolean type. Defines _NO_LIBGDSL_TYPES_ at compilation time if you don't want them.

Enumerator:

- ***FALSE*** FALSE boolean value
- ***TRUE*** TRUE boolean value

Definition at line 270 of file gdsl_types.h.

Chapter 5

File Documentation

5.1 _gdsl_bintree.h File Reference

Low level binary tree manipulation module.

Typedefs

- typedef struct _gdsl_bintree ∗ **_gdsl_bintree_t**

  GDSL low-level binary tree type.
- typedef int(∗ **_gdsl_bintree_map_func_t**)(const **_gdsl_bintree_t** TREE, void ∗USER_DATA)

  GDSL low-level binary tree map function type.
- typedef void(∗ **_gdsl_bintree_write_func_t**)(const **_gdsl_bintree_t** TREE, FILE ∗OUTPUT_FILE, void ∗USER_DATA)

  GDSL low-level binary tree write 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_bintree_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_bintree-_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_bintree_-write_func_t** WRITE_F, FILE ∗OUTPUT_FILE, void ∗USER_DATA)

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

5.1.1 Detailed Description

Low level binary tree manipulation module.

5.1.2 Copyright

Definition in file **_gdsl_bintree.h**.

5.2 _gdsl_bstree.h File Reference

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.

- typedef void(∗ **_gdsl_bstree_write_func_t**)(**_gdsl_bstree_t** TREE, FILE ∗OU-TPUT_FILE, void ∗USER_DATA)

GDSL low-level binary search tree write 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_bstree_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_bstree_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_bstree_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.

5.2.1 Detailed Description

Low level binary search tree manipulation module.

5.2.2 Copyright

Definition in file **_gdsl_bstree.h**.

5.3 _gdsl_list.h File Reference

Low-level doubly-linked list manipulation module.

Typedefs

- typedef **_gdsl_node_t _gdsl_list_t**

GDSL low-level doubly-linked list 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_node-_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_-node_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_node_write_func_t** -WRITE_F, FILE ∗OUTPUT_FILE, void ∗USER_DATA)

  Write all nodes of a low-level list to a file.
- void **_gdsl_list_write_xml** (const **_gdsl_list_t** L, const **_gdsl_node_write_-func_t** WRITE_F, FILE ∗OUTPUT_FILE, void ∗USER_DATA)

  Write all nodes of a low-level list to a file into XML.
- void **_gdsl_list_dump** (const **_gdsl_list_t** L, const **_gdsl_node_write_func_t** WRITE_F, FILE ∗OUTPUT_FILE, void ∗USER_DATA)

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

5.3.1 Detailed Description

Low-level doubly-linked list manipulation module.

5.3.2 Copyright

Definition in file **_gdsl_list.h**.

5.4 _gdsl_node.h File Reference

Low-level doubly-linked node manipulation module.

Typedefs

- typedef struct _gdsl_node ∗ **_gdsl_node_t**

  GDSL low-level doubly linked node type.
- typedef int(∗ **_gdsl_node_map_func_t**)(const **_gdsl_node_t** NODE, void ∗USER_DATA)

  GDSL low-level doubly-linked node map function type.
- typedef void(∗ **_gdsl_node_write_func_t**)(const **_gdsl_node_t** NODE, FILE ∗OUTPUT_FILE, void ∗USER_DATA)

  GDSL low-level doubly-linked node write function 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** SUC-C)

  Set the successor of a low-level node.

- void **_gdsl_node_set_pred** (**_gdsl_node_t** NODE, const **_gdsl_node_t** PRE-D)

  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_node_write-_func_t** WRITE_F, FILE ∗OUTPUT_FILE, void ∗USER_DATA)

  Write a low-level node to a file.

- void **_gdsl_node_write_xml** (const **_gdsl_node_t** NODE, const **_gdsl_node_-write_func_t** WRITE_F, FILE ∗OUTPUT_FILE, void ∗USER_DATA)

  Write a low-level node to a file into XML.

- void **_gdsl_node_dump** (const **_gdsl_node_t** NODE, const **_gdsl_node_write-_func_t** WRITE_F, FILE ∗OUTPUT_FILE, void ∗USER_DATA)

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

5.4.1 Detailed Description

Low-level doubly-linked node manipulation module.

5.4.2 Copyright

Definition in file **_gdsl_node.h**.

5.5 gdsl.h File Reference

Main GDSL header file. Include it in your source code to be able to use all GDSL modules.

Functions

- const char * **gdsl_get_version** (void)

  Get GDSL version number as a string.

5.5.1 Detailed Description

Main GDSL header file. Include it in your source code to be able to use all GDSL modules.

5.5.2 Copyright

This file is part of the Generic Data Structures Library (GDSL). Copyright (C) 1998-2018 Nicolas Darnis <`ndarnis@free.fr`>.

GDSL is free software: you can redistribute it and/or modify it under the terms of the GN-U General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

GDSL is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTI-CULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with GDSL. If not, see <`http://www.gnu.org/licenses/`>.

Definition in file **gdsl.h**.

5.6 gdsl_2darray.h File Reference

2D-Arrays manipulation module.

Typedefs

- typedef struct gdsl_2darray * **gdsl_2darray_t**

  GDSL 2D-array type.

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 ∗NE-W_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_write_func_t** W-RITE_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.

5.6.1 Detailed Description

2D-Arrays manipulation module.

5.6.2 Copyright

Definition in file **gdsl_2darray.h**.

5.7 gdsl_bstree.h File Reference

Binary search tree manipulation module.

Typedefs

- typedef struct 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.

5.7.1 Detailed Description

Binary search tree manipulation module.

5.7.2 Copyright

Definition in file **gdsl_bstree.h**.

5.8 gdsl_hash.h File Reference

Hashtable manipulation module.

Typedefs

- typedef struct hash_table * **gdsl_hash_t**

  GDSL hashtable type.
- typedef const char *(* **gdsl_key_func_t**)(void *VALUE)

  GDSL hashtable key function type.
- typedef **ulong**(* **gdsl_hash_func_t**)(const char *KEY)

  GDSL hashtable hash function type.

Functions

- **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_NAM-E)

  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** M-AP_F, void ∗USER_DATA)

  Parse a hashtable.
- void **gdsl_hash_write** (const **gdsl_hash_t** H, **gdsl_write_func_t** WRITE_F, FI-LE ∗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, F-ILE ∗OUTPUT_FILE, void ∗USER_DATA)

  Dump the internal structure of a hashtable to a file.

5.8.1 Detailed Description

Hashtable manipulation module.

5.8.2 Copyright

Definition in file **gdsl_hash.h**.

5.9 gdsl_heap.h File Reference

Heap manipulation module.

Typedefs

- typedef struct heap ∗ **gdsl_heap_t**

  GDSL heap type.

Functions

- **gdsl_heap_t gdsl_heap_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 heap.
- void **gdsl_heap_free** (**gdsl_heap_t** H)

  Destroy a heap.
- void **gdsl_heap_flush** (**gdsl_heap_t** H)

  Flush a heap.
- const char ∗ **gdsl_heap_get_name** (const **gdsl_heap_t** H)

  Get the name of a heap.
- **ulong gdsl_heap_get_size** (const **gdsl_heap_t** H)

  Get the size of a heap.
- **gdsl_element_t gdsl_heap_get_top** (const **gdsl_heap_t** H)

  Get the top of a heap.
- **bool gdsl_heap_is_empty** (const **gdsl_heap_t** H)

  Check if a heap is empty.
- **gdsl_heap_t gdsl_heap_set_name** (**gdsl_heap_t** H, const char ∗NEW_NAM-E)

  Set the name of a heap.
- **gdsl_element_t gdsl_heap_set_top** (**gdsl_heap_t** H, void ∗VALUE)

  Substitute the top element of a heap by a lesser one.
- **gdsl_element_t gdsl_heap_insert** (**gdsl_heap_t** H, void ∗VALUE)

  Insert an element into a heap (PUSH).
- **gdsl_element_t gdsl_heap_remove_top** (**gdsl_heap_t** H)

  Remove the top element from a heap (POP).
- **gdsl_heap_t gdsl_heap_delete_top** (**gdsl_heap_t** H)

  Delete the top element from a heap.
- **gdsl_element_t gdsl_heap_map_forward** (const **gdsl_heap_t** H, **gdsl_map_-func_t** MAP_F, void ∗USER_DATA)

  Parse a heap.
- void **gdsl_heap_write** (const **gdsl_heap_t** H, **gdsl_write_func_t** WRITE_F, FI-LE ∗OUTPUT_FILE, void ∗USER_DATA)

  Write all the elements of a heap to a file.

- void **gdsl_heap_write_xml** (const **gdsl_heap_t** H, **gdsl_write_func_t** WRITE-_F, FILE *OUTPUT_FILE, void *USER_DATA)

    Write the content of a heap to a file into XML.
- void **gdsl_heap_dump** (const **gdsl_heap_t** H, **gdsl_write_func_t** WRITE_F, F-ILE *OUTPUT_FILE, void *USER_DATA)

    Dump the internal structure of a heap to a file.

5.9.1 Detailed Description

Heap manipulation module.

5.9.2 Copyright

Definition in file **gdsl_heap.h**.

5.10 gdsl_interval_heap.h File Reference

Interval Heap manipulation module.

Typedefs

- typedef struct heap * **gdsl_interval_heap_t**

    GDSL interval heap type.

Functions

- **gdsl_interval_heap_t gdsl_interval_heap_alloc** (const char *NAME, **gdsl_-alloc_func_t** ALLOC_F, **gdsl_free_func_t** FREE_F, **gdsl_compare_func_t** C-OMP_F)

    Create a new interval heap.
- void **gdsl_interval_heap_free** (**gdsl_interval_heap_t** H)

Destroy an interval heap.

- void **gdsl_interval_heap_flush** (**gdsl_interval_heap_t** H)

  Flush an interval heap.
- const char ∗ **gdsl_interval_heap_get_name** (const **gdsl_interval_heap_t** H)

  Get the name of an interval heap.
- **ulong gdsl_interval_heap_get_size** (const **gdsl_interval_heap_t** H)

  Get the size of a interval heap.
- void **gdsl_interval_heap_set_max_size** (const **gdsl_interval_heap_t** H, **ulong** size)

  Set the maximum size of the interval heap.
- **bool gdsl_interval_heap_is_empty** (const **gdsl_interval_heap_t** H)

  Check if an interval heap is empty.
- **gdsl_interval_heap_t gdsl_interval_heap_set_name** (**gdsl_interval_heap_t** - H, const char ∗NEW_NAME)

  Set the name of an interval heap.
- **gdsl_element_t gdsl_interval_heap_insert** (**gdsl_interval_heap_t** H, void ∗V- ALUE)

  Insert an element into an interval heap (PUSH).
- **gdsl_element_t gdsl_interval_heap_remove_max** (**gdsl_interval_heap_t** H)

  Remove the maximum element from an interval heap (POP).
- **gdsl_element_t gdsl_interval_heap_remove_min** (**gdsl_interval_heap_t** - H)

  Remove the minimum element from an interval heap (POP).
- **gdsl_element_t gdsl_interval_heap_get_min** (const **gdsl_interval_heap_t** - H)

  Get the minimum element.
- **gdsl_element_t gdsl_interval_heap_get_max** (const **gdsl_interval_heap_t** - H)

  Get the maximum element.
- **gdsl_interval_heap_t gdsl_interval_heap_delete_min** (**gdsl_interval_heap_t** H)

  Delete the minimum element from an interval heap.
- **gdsl_interval_heap_t gdsl_interval_heap_delete_max** (**gdsl_interval_heap- _t** H)

  Delete the maximum element from an interval heap.
- **gdsl_element_t gdsl_interval_heap_map_forward** (const **gdsl_interval_- heap_t** H, **gdsl_map_func_t** MAP_F, void ∗USER_DATA)

  Parse a interval heap.
- void **gdsl_interval_heap_write** (const **gdsl_interval_heap_t** H, **gdsl_write_- func_t** WRITE_F, FILE ∗OUTPUT_FILE, void ∗USER_DATA)

  Write all the elements of an interval heap to a file.
- void **gdsl_interval_heap_write_xml** (const **gdsl_interval_heap_t** H, **gdsl_- write_func_t** WRITE_F, FILE ∗OUTPUT_FILE, void ∗USER_DATA)

  Write the content of an interval heap to a file into XML.

- void **gdsl_interval_heap_dump** (const **gdsl_interval_heap_t** H, **gdsl_write_-func_t** WRITE_F, FILE *OUTPUT_FILE, void *USER_DATA)

    Dump the internal structure of an interval heap to a file.

5.10.1 Detailed Description

Interval Heap manipulation module.

5.10.2 Copyright

Definition in file **gdsl_interval_heap.h**.

5.11 gdsl_list.h File Reference

Doubly-linked list manipulation module.

Typedefs

- typedef struct _gdsl_list * **gdsl_list_t**

    GDSL doubly-linked list type.
- typedef struct _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** CO-MP_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** P-OS)

  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)

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, F-ILE ∗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 ∗V-ALUE)
  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.

5.11.1 Detailed Description

Doubly-linked list manipulation module.

5.11.2 Copyright

This file is part of the Generic Data Structures Library (GDSL). Copyright (C) 1998-2018 Nicolas Darnis <ndarnis@free.fr>.

GDSL is free software: you can redistribute it and/or modify it under the terms of the GN-U General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

GDSL is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTI-CULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with GDSL. If not, see <http://www.gnu.org/licenses/>.

Definition in file **gdsl_list.h**.

5.12 gdsl_macros.h File Reference

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.

5.12.1 Detailed Description

Various macros module.

5.12.2 Copyright

Definition in file **gdsl_macros.h**.

5.13 gdsl_perm.h File Reference

Permutation manipulation module.

Typedefs

- typedef struct gdsl_perm ∗ **gdsl_perm_t**

  GDSL permutation type.
- typedef void(∗ **gdsl_perm_write_func_t**)(**ulong** E, FILE ∗OUTPUT_FILE, **gdsl-_location_t** POSITION, void ∗USER_DATA)

  GDSL permutation write function type.
- typedef struct gdsl_perm_data ∗ **gdsl_perm_data_t**

Enumerations

- enum **gdsl_perm_position_t** { **GDSL_PERM_POSITION_FIRST** = 1, **GDSL_P-ERM_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_NAM-E)

  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_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.

5.13.1 Detailed Description

Permutation manipulation module.

5.13.2 Copyright

Definition in file **gdsl_perm.h**.

5.14 gdsl_queue.h File Reference

Queue manipulation module.

Typedefs

- typedef struct _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.

5.14.1 Detailed Description

Queue manipulation module.

5.14.2 Copyright

Definition in file **gdsl_queue.h**.

5.15 gdsl_rbtree.h File Reference

Red-black tree manipulation module.

Typedefs

- typedef struct 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_NA-ME)

  Set the name of a red-black tree.
- **gdsl_element_t gdsl_rbtree_insert** (**gdsl_rbtree_t** T, void ∗VALUE, int ∗RES-ULT)

  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** WRIT-E_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.

5.15.1 Detailed Description

Red-black tree manipulation module.

5.15.2 Copyright

Definition in file **gdsl_rbtree.h**.

5.16 gdsl_sort.h File Reference

Sort module.

Functions

- void **gdsl_sort** (**gdsl_element_t** ∗T, **ulong** N, const **gdsl_compare_func_t** COMP_F)

  Sort an array in place.

5.16.1 Detailed Description

Sort module.

5.16.2 Copyright

Definition in file **gdsl_sort.h**.

5.17 gdsl_stack.h File Reference

Stack manipulation module.

Typedefs

- typedef struct _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.
- **ulong 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, **ulong** 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, F-ILE ∗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, FI-LE ∗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.

5.17.1 Detailed Description

Stack manipulation module.

5.17.2 Copyright

This file is part of the Generic Data Structures Library (GDSL). Copyright (C) 1998-2018 Nicolas Darnis <`ndarnis@free.fr`>.

GDSL is free software: you can redistribute it and/or modify it under the terms of the GN-U General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

GDSL is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTI-CULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with GDSL. If not, see <`http://www.gnu.org/licenses/`>.

Definition in file **gdsl_stack.h**.

5.18 gdsl_types.h File Reference

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, **gdsl_location_t** LOCATION, void ∗USER_DATA)

  GDSL Map element function type.
- typedef long 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, **gdsl_location_t** LOCATION, void ∗USER_DATA)

  GDSL Write element function type.
- typedef unsigned long int **ulong**
- typedef unsigned short int **ushort**

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 **gdsl_location_t** { **GDSL_LOCATION_UNDEF** = 0, **GDSL_LOCATION_HEAD** = 1, **GDSL_LOCATION_ROOT** = 1, **GDSL_LOCATION_TOP** = 1, × **GDSL_LOCATION_TAIL** = 2, **GDSL_LOCATION_LEAF** = 2, **GDSL_LOCATION_BOTTOM** = 2, **GDSL_LOCATION_FIRST** = 1, **GDSL_LOCATION_LAST** = 2, **GDSL_LOCATION_FIRST_COL** = 1, **GDSL_LOCATION_LAST_COL** = 2, **GDSL_LOCATION_FIRST_ROW** = 4, **GDSL_LOCATION_LAST_ROW** = 8 }
- enum **bool** { **FALSE** = 0, **TRUE** = 1 }

5.18.1 Detailed Description

GDSL types.

5.18.2 Copyright

This file is part of the Generic Data Structures Library (GDSL). Copyright (C) 1998-2018 Nicolas Darnis <ndarnis@free.fr>.

GDSL is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

GDSL is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with GDSL. If not, see <http://www.gnu.org/licenses/>.

Definition in file **gdsl_types.h**.

5.19 mainpage.h File Reference

Chapter 6

Example Documentation

6.1 examples/main_2darray.c

This is an example of how to use gdsl_2darray module.

```
#include <config.h>


#include <stdio.h>
#include <stdlib.h>
#include <string.h>


#include "gdsl_2darray.h"
#include "_integers.h"


#define ROWS_NB 4UL
#define COLS_NB 3UL


static void
my_display_integer (const gdsl_element_t e, FILE* file, gdsl_location_t
      position, void* user_data)
{
    int* n = (int*) e;

    if (position & GDSL_LOCATION_FIRST_COL)
    {
        if (position & GDSL_LOCATION_FIRST_ROW)
        {
            fprintf (file, "{\n");
        }

        fprintf (file, "\t( ");
    }
    else
    {
        fprintf (file, " ");
    }
```

```
    fprintf (file, "%02d", *n);

    if (position & GDSL_LOCATION_LAST_COL)
    {
        fprintf (file, " )\n");

        if (position & GDSL_LOCATION_LAST_ROW)
        {
            fprintf (file, "\t}\n");
        }
    }
}

int main (void)
{
    int i, j, k = 0;
    gdsl_2darray_t m = gdsl_2darray_alloc ("MY ARRAY", ROWS_NB, COLS_NB,
      alloc_integer, free_integer);

    for (i = 0; i < ROWS_NB; i++)
    {
        for (j = 0; j < COLS_NB; j++)
        {
            int n = ++k;
            gdsl_2darray_set_content (m, i, j, (void*) &n);
        }
    }

    printf ("%s (%ld x %ld) = ", gdsl_2darray_get_name (m),
        gdsl_2darray_get_rows_number (m),
        gdsl_2darray_get_columns_number (m));

    gdsl_2darray_write (m, my_display_integer, stdout, NULL);
    gdsl_2darray_write_xml (m, my_display_integer, stdout, NULL);
    gdsl_2darray_dump (m, my_display_integer, stdout, NULL);

    gdsl_2darray_free (m);

    exit (EXIT_SUCCESS);
}
```

6.2 examples/main_bstree.c

This is an example of how to use gdsl_bstree module.

```
#include <config.h>


#include <stdio.h>
#include <string.h>
#include <stdlib.h>


#include "gdsl_perm.h"
#include "gdsl_bstree.h"
```

```
#include "_strings.h"
#include "_integers.h"


#define N 100


int main (int argc, char *argv[])
{
    int choice;
    char name[50];
    gdsl_bstree_t t = gdsl_bstree_alloc ("MY BSTREE", alloc_string, free_string
      , compare_strings);

    do
    {
        printf ("\t\tMENU - BSTREE\n\n");
        printf ("\t 1> Insert\n");
        printf ("\t 2> Remove\n");
        printf ("\t 3> Flush\n");
        printf ("\t 4> Root content\n");
        printf ("\t 5> Size\n");
        printf ("\t 6> Height\n");
        printf ("\t 7> Search\n");
        printf ("\t 8> Display\n");
        printf ("\t 9> XML display\n");
        printf ("\t10> Dump\n");
        printf ("\t11> Insertion of a random permutation\n");
        printf ("\t 0> Quit\n\n");
        printf ("\t\tYour choice: ");
        scanf ("%d", &choice);

        switch (choice)
        {
        case 1:
            {
            int rc;

            printf ("Enter a string: ");
            scanf ("%s", name);

            gdsl_bstree_insert (t, (void*) name, &rc);

            if (rc == GDSL_FOUND)
                {
                printf ("'%s' is already into the tree\n", name);
                }
            else if (rc == GDSL_ERR_MEM_ALLOC)
                {
                printf ("memory allocation error\n");
                }
            }
            break;

        case 2:
            if (gdsl_bstree_is_empty (t))
            {
                printf ("The tree is empty.\n");
            }
            else
            {
                printf ("Enter a string: ");
```

```
            scanf ("%s", name);

            if (gdsl_bstree_delete (t, (void *) name))
            {
                printf ("String '%s' removed from the tree\n", name);
            }
            else
            {
                printf ("String '%s' not found\n", name);
            }
        }
        break;

    case 3:
        gdsl_bstree_flush (t);
        break;

    case 4:
        if (gdsl_bstree_is_empty (t))
        {
            printf ("The tree is empty.\n");
        }
        else
        {
            print_string (gdsl_bstree_get_root (t), stdout,
  GDSL_LOCATION_UNDEF, " \n");
        }
        break;

    case 5:
        printf ("Tree's size: %lu\n", gdsl_bstree_get_size (t));
        break;

    case 6:
        printf ("Tree's height: %lu\n", gdsl_bstree_get_height (t));
        break;

    case 7:
        printf ("Enter a string: ");
        scanf ("%s", name);

        if (gdsl_bstree_search (t, NULL, (void *) name))
        {
            printf ("String '%s' found\n", name);
        }
        else
        {
            printf ("String '%s' not found\n", name);
        }
        break;

    case 8:
        if (gdsl_bstree_is_empty (t))
        {
            printf ("The tree is empty.\n");
        }
        else
        {
            printf ("Tree's content: ");
            gdsl_bstree_write (t, print_string, stdout, NULL);
            printf ("\n");
        }
```

```
            break;

        case 9:
            gdsl_bstree_write_xml (t, print_string, stdout, NULL);
            break;

        case 10:
            gdsl_bstree_dump (t, print_string, stdout, NULL);
            break;

        case 11:
            {
            int i;
            int rc;
            gdsl_perm_t   p = gdsl_perm_alloc ("p", N);
            gdsl_bstree_t nt = gdsl_bstree_alloc ("INTEGERS", alloc_integer,
      free_integer, compare_integers);

            gdsl_perm_randomize (p);

            for (i = 0; i < N; i++)
                {
                int n = gdsl_perm_get_element (p, i);
                gdsl_bstree_insert (nt, &n, &rc);
                }

            printf ("Tree's height: %lu\n", gdsl_bstree_get_height (nt));
            gdsl_bstree_dump (nt, print_integer, stdout, (void*) "");

            gdsl_bstree_free (nt);
            gdsl_perm_free (p);
            }
            break;
        }
    }
    while (choice != 0);

    gdsl_bstree_free (t);

    exit (EXIT_SUCCESS);
}
```

6.3 examples/main_hash.c

This is an example of how to use gdsl_hash module.

```
#include <config.h>


#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <mcheck.h>


#include "gdsl_types.h"
```

```
#include "gdsl_hash.h"
#include "_strings.h"


#define SIZE 11 /* Should be prime number !! */


struct _my_struct
{
    int  integer;
    char*string;
};
typedef struct _my_struct* my_struct;

static gdsl_element_t
my_struct_alloc (void* d)
{
    static int n = 0;

    my_struct e = (my_struct) malloc (sizeof (struct _my_struct));
    if (e == NULL)
    {
        return NULL;
    }

    e->integer = n++;
    e->string = strdup ((char*) d);

    return (gdsl_element_t) e;
}

static void
my_struct_free (gdsl_element_t e)
{
    my_struct s = (my_struct) e;
    free (s->string);
    free (s);
}

static void
my_struct_printf (gdsl_element_t e, FILE* file, gdsl_location_t location, void*
       d)
{
    my_struct s = (my_struct) e;
    fprintf (file, "%d:%s ", s->integer, s->string);
}

const char*
my_struct_key (gdsl_element_t e)
{
    my_struct s = (my_struct) e;
    return s->string;
}

int main (void)
{
    int         choice;
    gdsl_hash_t ht;

    mtrace ();

    ht = gdsl_hash_alloc ("MY HASH TABLE", my_struct_alloc, my_struct_free,
```

```
  my_struct_key, NULL, SIZE);
if (ht == NULL)
{
    fprintf (stderr, "%s:%d: %s - gdsl_hash_alloc(): NULL",
         __FILE__, __LINE__, __FUNCTION__);
    exit (EXIT_FAILURE);
}

do
{
    printf ("\t\tMENU - HASH\n\n");
    printf ("\t1> Insert\n");
    printf ("\t2> Search\n");
    printf ("\t3> Remove\n");
    printf ("\t4> Display\n");
    printf ("\t5> Flush\n");
    printf ("\t6> Fill factor\n");
    printf ("\t7> Dump\n");
    printf ("\t8> XML display\n");
    printf ("\t0> Quit\n\n");
    printf ("\t\tYour choice: ");
    scanf ("%d", &choice);

    switch (choice)
    {
    case 1:
        {
        char nom[50];

        printf ("String: ");
        scanf ("%s", nom);

        if (gdsl_hash_insert (ht, (void*) nom) == NULL)
            {
            printf ("ERROR: Insert failed!\n");
            }
        }
        break;

    case 2:
        {
        char nom[50];
        gdsl_element_t e;

        printf ("String: ");
        scanf ("%s", nom);

        e = gdsl_hash_search (ht, nom);
        if (e == NULL)
            {
            printf ("String '%s' doesn't exist\n", nom);
            }
        else
            {
            printf ("String '%s' found\n", nom);
            }
        }
        break;

    case 3:
        {
        char nom[50];
```

```
            gdsl_element_t e;

            printf ("String: ");
            scanf ("%s", nom);

            e = gdsl_hash_remove (ht, nom);
            if (e == NULL)
                {
                printf ("String '%s' doesn't exist\n", nom);
                }
            else
                {
                free_string (e);
                }
            }
            break;

        case 4:
            gdsl_hash_write (ht, my_struct_printf, stdout, " ");
            printf ("\n");
            break;

        case 5:
            gdsl_hash_flush (ht);
            break;

        case 6:
            printf ("Fill factor: %g\n", gdsl_hash_get_fill_factor (ht));
            break;

        case 7:
            gdsl_hash_dump (ht, my_struct_printf, stdout, NULL);
            break;

        case 8:
            gdsl_hash_write_xml (ht, my_struct_printf, stdout, NULL);
            break;
        }
    }
    while (choice != 0);

    gdsl_hash_free (ht);

    exit (EXIT_SUCCESS);
}
```

6.4 examples/main_heap.c

This is an example of how to use gdsl_heap module.

```
#include <config.h>


#include <stdio.h>
#include <stdlib.h>
#include <string.h>
```

```
#include "gdsl_types.h"
#include "gdsl_heap.h"


#include "_integers.h"


static int
my_display_integer (const gdsl_element_t e, gdsl_location_t location,
            void* user_infos)
{
    printf ("%s%s%ld ",
        (location & GDSL_LOCATION_ROOT) ? "[root]: " : "",
        (location & GDSL_LOCATION_LEAF) ? "[leaf]: " : "",
        *(long int*) e);
    return GDSL_MAP_CONT;
}

int main (void)
{
    int choix = 0;
    gdsl_heap_t h = gdsl_heap_alloc ("H", alloc_integer, free_integer,
      compare_integers);

    do
    {
        printf ("\t\tMENU - HEAP\n\n");
        printf ("\t1> Push: insert an element\n");
        printf ("\t2> Pop: remove max element\n");
        printf ("\t3> Get: peek max element\n");
        printf ("\t4> Set: substitute max element\n");
        printf ("\t5> Flush\n");
        printf ("\t6> Remove: *** NOT YET IMPLEMENTED ***\n");
        printf ("\t7> Display\n");
        printf ("\t8> Dump\n");
        printf ("\t9> XML display\n");
        printf ("\t0> Quit\n\n");
        printf ("\t\tYour choice: ");
        scanf ("%d", &choix);

        switch (choix)
        {
        case 1:
            {
            int value;

            printf ("Enter integer value: ");
            scanf ("%d", &value);
            gdsl_heap_insert (h, (void*) &value);
            }
            break;

        case 2:
            if (!gdsl_heap_is_empty (h))
            {
                gdsl_heap_delete_top (h);
            }
            else
            {
                printf ("The heap '%s' is empty\n", gdsl_heap_get_name (h));
```

```
        }
        break;

    case 3:
        {
        long int* top;

        if (!gdsl_heap_is_empty (h))
            {
            top = (long int*) gdsl_heap_get_top (h);
            printf ("Value = %ld\n", *top);
            }
        else
            {
            printf ("The heap '%s' is empty\n", gdsl_heap_get_name (h));
            }
        }
        break;

    case 4:
        {
        int value;
        long int* v;

        printf ("Enter integer value: ");
        scanf ("%d", &value);
        v = (long int*) gdsl_heap_set_top (h, (void*) &value);
        if (v == NULL)
            {
            printf ("value is greather than all other heap ones\n");
            }
        else
            {
            printf ("old value was: %ld\n", *v);
            free_integer (v);
            }
        }
        break;

    case 5:
        if (gdsl_heap_is_empty (h))
        {
            printf ("The heap '%s' is empty\n", gdsl_heap_get_name (h));
        }
        else
        {
            gdsl_heap_flush (h);
        }
        break;
        /*
    case 6:
        {
        int pos;
        long int* value;
        printf ("Enter an integer value to search: ");
        scanf ("%d", &pos);

        value = (long int*) gdsl_heap_remove (h, &pos);
        if (value == NULL)
            {
            printf ("Not found\n");
            }
```

```
            else
                {
                printf ("Value removed %ld\n", *value);
                free_integer (value);
                }
            }
            break;
            */
        case 7:
            if (gdsl_heap_is_empty (h))
            {
                printf ("The heap '%s' is empty\n", gdsl_heap_get_name (h));
            }
            else
            {
                printf ("%s = ( ", gdsl_heap_get_name (h));
                gdsl_heap_map_forward (h, my_display_integer, NULL);
                printf (")\n");
            }
            break;

        case 8:
            gdsl_heap_dump (h, print_integer, stdout, NULL);
            break;

        case 9:
            gdsl_heap_write_xml (h, print_integer, stdout, NULL);
            break;
        }
    }
    while (choix != 0);

    gdsl_heap_free (h);

    exit (EXIT_SUCCESS);
}
```

6.5 examples/main_interval_heap.c

This is an example of how to use gdsl_interval_heap module.

```
#include <config.h>

#include <assert.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>


#include "gdsl_types.h"
#include "gdsl_interval_heap.h"


#include "_integers.h"
```

```
static int
my_display_integer (const gdsl_element_t e, gdsl_location_t location,
            void* user_infos)
{
    //printf ("user_info: %d\n", *(int *)user_infos);
    printf ("%s%s%ld ",
        (location & GDSL_LOCATION_ROOT) ? "[root]: " : "",
        (location & GDSL_LOCATION_LEAF) ? "[leaf]: " : "",
        *(long int*) e);
    return GDSL_MAP_CONT;
}

void insert_value(gdsl_interval_heap_t h, long int a) {
        //int value = rand() % 20;
        long int *value = malloc(sizeof(int));
        //*value = rand() % 20;
        *value = a;
        //printf("inserting value: %d\n", *value);
        gdsl_interval_heap_insert(h, (void *) value);
        //gdsl_raw_heap_dump(h);
        //gdsl_check_interval_heap_integrity(h);
}

long *remove_max(gdsl_interval_heap_t h) {
    long *value;
    //printf("removing max\n");
    //gdsl_raw_heap_dump(h);
    value = gdsl_interval_heap_remove_max(h);
    //printf("removed value: %x %d\n", value, *value);
    //gdsl_raw_heap_dump(h);
    //gdsl_check_interval_heap_integrity(h);

    return value;
}

long *remove_min(gdsl_interval_heap_t h) {
    long *value;
    //printf("removing min\n");
    //gdsl_raw_heap_dump(h);
    value = gdsl_interval_heap_remove_min(h);
    //printf("removed value: %x %d\n", value, *value);
    //gdsl_raw_heap_dump(h);
    //gdsl_check_interval_heap_integrity(h);

    return value;
}

void test1() {
    gdsl_interval_heap_t h = gdsl_interval_heap_alloc ("H", alloc_integer,
      free_integer, compare_integers);

    insert_value(h, 2);
    insert_value(h, 30);
    insert_value(h, 3);
    insert_value(h, 20);
    insert_value(h, 4);
    insert_value(h, 25);
    insert_value(h, 8);
    insert_value(h, 16);
    insert_value(h, 4);
    //exit(0);
    insert_value(h, 10);
```

```
    insert_value(h, 10);
    insert_value(h, 15);
    insert_value(h, 5);
    insert_value(h, 12);
    insert_value(h, 8);
    insert_value(h, 16);
    insert_value(h, 9);
    insert_value(h, 15);
    insert_value(h, 5);

    insert_value(h, 1);
    insert_value(h, 25);

    remove_min(h);

    remove_max(h);

    remove_min(h);
    remove_min(h);
    remove_min(h);
    remove_min(h);

    gdsl_interval_heap_flush(h);

    assert(gdsl_interval_heap_get_size(h) == 0);

    gdsl_interval_heap_free (h);

}

void test2() {
    gdsl_interval_heap_t h = gdsl_interval_heap_alloc ("H", alloc_integer,
      free_integer, compare_integers);

    gdsl_interval_heap_flush(h);

    insert_value(h, 2);
    insert_value(h, 2);
    insert_value(h, 2);

    //remove_min(h);
    remove_max(h);
    remove_max(h);
    remove_max(h);
    //remove_min(h);
    //remove_min(h);

    assert(gdsl_interval_heap_get_size(h) == 0);

    gdsl_interval_heap_free (h);
}

void check_removed(long **removed, int len) {
    int i, j;
    for (i = 0; i < len; i++) {
        for (j = i+1; j < len; j++) {
            assert(removed[i] != removed[j]);
        }
    }

    //printf("checked removed\n");
}
```

```
void test3() {
    int i, len=2000;
    long *e;
    gdsl_interval_heap_t h = gdsl_interval_heap_alloc ("H", alloc_integer,
      free_integer, compare_integers);
    gdsl_interval_heap_flush(h);
    long **removed = malloc(len * sizeof(long *));

    for (i = 0; i < len; i++) {
        insert_value(h, rand() % 20);
    }

    for (i = 0; i < len/2; i++) {
        if (i % 2 == 0)
            e = remove_min(h);
        else
            e = remove_max(h);

        removed[i] = e;
    }

    check_removed(removed, len/2);

    for (i = 0; i < len/2; i++) {
        insert_value(h, rand() % 20);
    }

    for (i = 0; i < len; i++) {
        if (i % 2 == 0)
            e = remove_min(h);
        else
            e = remove_max(h);

        removed[i] = e;
    }

    check_removed(removed, len/2);

    assert(gdsl_interval_heap_get_size(h) == 0);
    gdsl_interval_heap_free (h);
}

int test4() {
    int i = 0, len = 20000;

    gdsl_interval_heap_t h = gdsl_interval_heap_alloc ("H", alloc_integer,
      free_integer, compare_integers);
    gdsl_interval_heap_flush(h);

    for (i = 0; i < len; i++) {
        if (rand() % 2 == 0) {
            insert_value(h, rand() % 40);
        } else {
            if (gdsl_interval_heap_get_size(h) > 1) {
                if (rand() % 2 == 0)
                    remove_min(h);
                else
                    remove_max(h);
            }

        }
```

```
    }

    //assert(gdsl_interval_heap_get_size(h) == 0);
    gdsl_interval_heap_free (h);
}

int main (void)
{
    //test2();
    //test3();
    test4();
    exit (EXIT_SUCCESS);
}
```

6.6 examples/main_list.c

This is an example of how to use gdsl_list module.

```
#include <config.h>


#include <stdio.h>
#include <stdlib.h>
#include <string.h>


#include "gdsl_perm.h"
#include "gdsl_types.h"
#include "gdsl_list.h"
#include "_strings.h"
#include "_integers.h"


#define PERMUTATION_NB 26


static void
affiche_liste_chaines_fwd (gdsl_list_t l)
{
    gdsl_element_t e;
    gdsl_list_cursor_t c = gdsl_list_cursor_alloc (l);

    printf ("%s (->) = ( ", gdsl_list_get_name (l));

    for (gdsl_list_cursor_move_to_head (c); (e = gdsl_list_cursor_get_content (
      c)); gdsl_list_cursor_step_forward (c))
    {
        print_string (e, stdout, GDSL_LOCATION_UNDEF, (void*) " ");
    }

    printf (")\n");

    gdsl_list_cursor_free (c);
}
```

```
static int
my_display_string (gdsl_element_t e, gdsl_location_t location, void* d)
{
    print_string (e, stdout, location, d);
    return GDSL_MAP_CONT;
}

static void
affiche_liste_chaines_bwd (gdsl_list_t l)
{
    printf ("%s (<-) = ( ", gdsl_list_get_name (l));

    gdsl_list_map_backward (l, my_display_string, (void*) " ");

    printf (")\n");
}

int main (void)
{
    int choix = 0;

    gdsl_list_t l = gdsl_list_alloc ("MY LIST", alloc_string, free_string);

    do
    {
        printf ("\t\tMENU - LIST\n\n");
        printf ("\t 1> Create a cell\n");
        printf ("\t 2> Remove the first cell\n");
        printf ("\t 3> Remove the last cell\n");
        printf ("\t 4> Remove a cell\n");
        printf ("\t 5> Display list in forward order\n");
        printf ("\t 6> Display list in backward order\n");
        printf ("\t 7> Flush list\n");
        printf ("\t 8> Size of list\n");
        printf ("\t 9> Dump list\n");
        printf ("\t10> XML dump of list\n");
        printf ("\t11> Search for a place\n");
        printf ("\t12> Search for an element\n");
        printf ("\t13> Sort of list\n");
        printf ("\t14> Greatest element of list\n");
        printf ("\t 0> Quit\n\n");
        printf ("\t\tYour choice: ");
        scanf ("%d", &choix);

        switch (choix)
        {
        case 1:
            {
            char nom[100];
            int done = 0;

            printf ("Nom: ");
            scanf ("%s", nom);

            do
                {
                int choix;

                printf ("\t\tMENU - CELL INSERTION\n\n");
                printf ("\t1> Insert cell at the beginning of the list\n");
                printf ("\t2> Insert cell at end of list\n");
                printf ("\t3> Insert cell after another cell\n");
```

```
            printf ("\t4> Insert cell before another cell\n");
            printf ("\t5> Display the list\n");
            printf ("\t0> RETURN TO MAIN MENU\n\n");
            printf ("\t\tYour choice: ");
            scanf ("%d", &choix );

            switch (choix)
                {
                case 1:
                {
                    gdsl_list_insert_head (l, nom);
                    done = 1;
                }
                break;

                case 2:
                {
                    gdsl_list_insert_tail (l, nom);
                    done = 1;
                }
                break;

                case 3:
                if (gdsl_list_is_empty (l))
                    {
                    printf ("The list is empty.\n");
                    }
                else
                    {
                    char Nom[100];
                    gdsl_list_cursor_t c = gdsl_list_cursor_alloc (l);

                    printf ("Name of cell after which you want to insert: "
);
                    scanf ("%s", Nom);

                    if (!gdsl_list_cursor_move_to_value (c, compare_strings
, Nom))
                        {
                        printf ("The cell '%s' doesn't exist\n", Nom);
                        }
                    else
                        {
                        gdsl_list_cursor_insert_after (c, nom);
                        done = 1;
                        }
                    gdsl_list_cursor_free (c);
                    }
                break;

                case 4:
                if (gdsl_list_is_empty (l))
                    {
                    printf ("The list is empty.\n");
                    }
                else
                    {
                    char Nom[100];
                    gdsl_list_cursor_t c = gdsl_list_cursor_alloc (l);

                    printf ("Name of cell before which you want to insert:
");
```

```
                    scanf ("%s", Nom);

                    if (!gdsl_list_cursor_move_to_value (c, compare_strings
, Nom))
                        {
                        printf ("The cell '%s' doesn't exist\n", Nom);
                        }
                    else
                        {
                        gdsl_list_cursor_insert_before (c, nom);
                        done = 1;
                        }
                    gdsl_list_cursor_free (c);
                    }
                break;

                case 5:
                if (gdsl_list_is_empty (l))
                    {
                    printf ("The list is empty.\n");
                    }
                else
                    {
                    affiche_liste_chaines_fwd (l);
                    }
                break;

                case 0:
                done = 1;
                break;
                }
            }
        while (!done);
        }
        break;

    case 2:
        if (gdsl_list_is_empty (l))
        {
            printf ("The list is empty.\n");
        }
        else
        {
            gdsl_list_delete_head (l);
        }
        break;

    case 3:
        if (gdsl_list_is_empty (l))
        {
            printf ("The list is empty.\n");
        }
        else
        {
            gdsl_list_delete_tail (l);
        }
        break;

    case 4:
        {
        char nom[100];
```

```
            if (gdsl_list_is_empty (l))
                {
                printf ("The list is empty.\n");
                }
            else
                {
                printf ("Name of cell to remove: ");
                scanf ("%s", nom);

                if (!gdsl_list_delete (l, compare_strings, nom))
                    {
                    printf ("The cell '%s' doesn't exist\n", nom);
                    }
                else
                    {
                    printf ("The cell '%s' is removed from list\n", nom);
                    }
                }
            }
            break;

        case 5:
            if (gdsl_list_is_empty (l))
            {
                printf ("The list is empty.\n");
            }
            else
            {
                affiche_liste_chaines_fwd (l);
            }
            break;

        case 6:
            if (gdsl_list_is_empty (l))
            {
                printf ("The list is empty.\n");
            }
            else
            {
                affiche_liste_chaines_bwd (l);
            }
            break;

        case 7:
            if (gdsl_list_is_empty (l))
            {
                printf ("The list is empty.\n");
            }
            else
            {
                gdsl_list_flush (l);
            }
            break;

        case 8:
            printf ("Card( %s ) = %ld\n", gdsl_list_get_name (l),
      gdsl_list_get_size (l));
            break;

        case 9:
            if (gdsl_list_is_empty (l))
            {
```

```
            printf ("The list is empty.\n");
        }
        else
        {
            gdsl_list_dump (l, print_string, stdout, NULL);
        }
        break;

    case 10:
        if (gdsl_list_is_empty (l))
        {
            printf ("The list is empty.\n");
        }
        else
        {
            gdsl_list_write_xml (l, print_string, stdout, NULL);
        }
        break;

    case 11:
        {
        int pos;
        gdsl_element_t e;

        printf ("Enter the position of the place to search for: ");
        scanf ("%d", & pos);

        e = gdsl_list_search_by_position (l, (ulong) pos);
        if (e != NULL)
            {
            print_string (e, stdout, GDSL_LOCATION_UNDEF, NULL);
            }
        }
        break;

    case 12:
        {
        char nom [100];
        gdsl_element_t e;

        printf ("Name of cell to search for: ");
        scanf ("%s", nom);

        e = gdsl_list_search (l, compare_strings, nom);
        if (e == NULL)
            {
            printf ("The cell '%s' doesn't exist\n", nom);
            }
        else
            {
            printf ("The cell '%s' was found: ", nom);
            print_string (e, stdout, GDSL_LOCATION_UNDEF, NULL);
            printf ("\n");
            }
        }
        break;

    case 13:
        gdsl_list_sort (l, compare_strings);
        break;

    case 14:
```

```
        if (gdsl_list_is_empty (l))
        {
            printf ("The list is empty.\n");
        }
        else
        {
            printf ("Max Element: %s\n", (char*) gdsl_list_search_max (l,
  compare_strings));
        }
        break;

    case 15: /* case for my own tests... */
        {
        int i;
        gdsl_perm_t p = gdsl_perm_alloc ("p", PERMUTATION_NB);
        gdsl_list_t g = gdsl_list_alloc ("MY LIST 2", alloc_string,
  free_string);

        gdsl_perm_randomize (p);

        for (i = 0; i < PERMUTATION_NB; i++)
            {
            char c[2];
            c[0] = 65 + gdsl_perm_get_element (p, i);
            c[1] = '\0';
            gdsl_list_insert_tail (g, c);
            }

        gdsl_perm_free (p);
        affiche_liste_chaines_fwd (g);
        affiche_liste_chaines_bwd (g);
        printf ("SORT\n");
        gdsl_list_sort (g, compare_strings);
        affiche_liste_chaines_fwd (g);
        affiche_liste_chaines_bwd (g);
        gdsl_list_free (g);
        }

        {
        int i = 0;
        gdsl_list_cursor_t c = gdsl_list_cursor_alloc (l);

        for (gdsl_list_cursor_move_to_head (c); gdsl_list_cursor_get_content
   (c); gdsl_list_cursor_step_forward (c))
            {
            char toto[50];
            sprintf (toto, "%d", i++);

            gdsl_list_cursor_insert_before (c, toto);

            gdsl_list_cursor_step_backward (c);
            gdsl_list_cursor_delete_after (c);
            }

        gdsl_list_cursor_free (c);
        }
        break;
    }
}
while (choix != 0);

gdsl_list_free (l);
```

```
    exit (EXIT_SUCCESS);
}
```

6.7 examples/main_llbintree.c

This is an example of how to use _gdsl_bintree module.

```
#include <config.h>


#include <stdio.h>
#include <string.h>
#include <stdlib.h>


#include "_gdsl_bintree.h"
#include "_strings.h"


static void
my_write_string (const _gdsl_bintree_t tree, FILE* file, void* d)
{
    gdsl_element_t e = _gdsl_bintree_get_content (tree);

    if (d == NULL)
    {
        fprintf (file, "%s", (char*) e);
    }
    else
    {
        fprintf (file, "%s%s", (char*) e, (char*) d);
    }
}

static int
my_map_string (const _gdsl_bintree_t t, void* d)
{
    my_write_string (t, stdout, d);
    return GDSL_MAP_CONT;
}

int main (void)
{
    _gdsl_bintree_t g, d, t, t1, t2, copy;

    g  = _gdsl_bintree_alloc ((gdsl_element_t) "b", NULL, NULL);
    d  = _gdsl_bintree_alloc ((gdsl_element_t) "o", NULL, NULL);
    t1 = _gdsl_bintree_alloc ((gdsl_element_t) "n", g, d);
    g  = _gdsl_bintree_alloc ((gdsl_element_t) "j", NULL, NULL);
    d  = _gdsl_bintree_alloc ((gdsl_element_t) "o", NULL, NULL);
    t2 = _gdsl_bintree_alloc ((gdsl_element_t) "u", g, d);
    t  = _gdsl_bintree_alloc ((gdsl_element_t) "r", t1, t2);

    printf ("T:\n");
    _gdsl_bintree_write_xml (t, my_write_string, stdout, NULL);
```

```
    copy = _gdsl_bintree_copy (t, copy_string);
    printf ("COPY OF T: \n");
    _gdsl_bintree_dump (copy, my_write_string, stdout, NULL);

    _gdsl_bintree_rotate_left (&t);
    _gdsl_bintree_rotate_right (&t);
    _gdsl_bintree_rotate_right (&t);
    _gdsl_bintree_rotate_left (&t);

    printf ("\nT in prefixed order: ");
    _gdsl_bintree_map_prefix (t, my_map_string, (void*) " ");
    printf ("\nT in infixed order: ");
    _gdsl_bintree_map_infix (t, my_map_string, (void*) " ");
    printf ("\nT in postfixed order: ");
    _gdsl_bintree_map_postfix (t, my_map_string, (void*) " ");

    printf ("\n\nCOPY OF T in prefixed order: ");
    _gdsl_bintree_map_prefix (copy, my_map_string, (void*) " ");
    printf ("\nCOPY OF T in infixed order: ");
    _gdsl_bintree_map_infix (copy, my_map_string, (void*) " ");
    printf ("\nCOPY OF T in postfixed order: ");
    _gdsl_bintree_map_postfix (copy, my_map_string, (void*) " ");
    printf ("\n\n");

    _gdsl_bintree_free (copy, free_string);
    _gdsl_bintree_free (t, NULL);

    exit (EXIT_SUCCESS);
}
```

6.8 examples/main_llbstree.c

This is an example of how to use _gdsl_bstree module.

```
#include <config.h>


#include <stdio.h>
#include <string.h>
#include <stdlib.h>


#include "gdsl_perm.h"
#include "_gdsl_bstree.h"
#include "_integers.h"
#include "_strings.h"


#define N 100

static void
my_write_string (const _gdsl_bstree_t tree, FILE* file, void* d)
{
    gdsl_element_t e = _gdsl_bstree_get_content (tree);
```

```
    if (d == NULL)
    {
        fprintf (file, "%s", (char*) e);
    }
    else
    {
        fprintf (file, "%s%s", (char*) e, (char*) d);
    }
}

static void
my_write_integer (const _gdsl_bstree_t tree, FILE* file, void* d)
{
    gdsl_element_t e = _gdsl_bstree_get_content (tree);
    long int** n = (long int**) e;

    if (d == NULL)
    {
        fprintf (file, "%ld", (long int) *n);
    }
    else
    {
        fprintf (file, "%ld%s", (long int) *n, (char*) d);
    }
}

int main (void)
{
    int rc;
    _gdsl_bstree_t t;

    printf ("Inserting 'a' in T... ");
    t  = _gdsl_bstree_alloc ((gdsl_element_t) "a");
    if (t != NULL)
    {
        printf ("OK\n");
    }

    printf ("Inserting 'b' in T... ");
    _gdsl_bstree_insert (&t, compare_strings, "b", &rc);
    if (rc == 0)
    {
        printf ("OK\n");
    }

    /* Volountary insertion of an existing element: */
    printf ("Inserting ALREADY EXISTING 'a' in T... ");
    _gdsl_bstree_insert (&t, compare_strings, "a", &rc);
    if (rc == GDSL_FOUND)
    {
        printf ("KO: a already exists in T\n");
    }

    printf ("Inserting 'c' in T... ");
    _gdsl_bstree_insert (&t, compare_strings, "c", &rc);
    if (rc == 0)
    {
        printf ("OK\n");
    }

    printf ("Inserting 'd' in T... ");
    _gdsl_bstree_insert (&t, compare_strings, "d", &rc);
```

```
    if (rc == 0)
    {
        printf ("OK\n");
    }

    printf ("Inserting 'e' in T... ");
    _gdsl_bstree_insert (&t, compare_strings, "e", &rc);
    if (rc == 0)
    {
        printf ("OK\n");
    }

    printf ("Inserting 'f' in T... ");
    _gdsl_bstree_insert (&t, compare_strings, "f", &rc);
    if (rc == 0)
    {
        printf ("OK\n");
    }

    printf ("T:\n");

    _gdsl_bstree_write_xml (t, my_write_string, stdout, NULL);
    _gdsl_bstree_free (t, NULL);

    {
    int i;
    gdsl_perm_t p = gdsl_perm_alloc ("p", N);
    _gdsl_bstree_t t = NULL;

    gdsl_perm_randomize (p);

    for (i = 0; i < N; i++)
        {
        int n = gdsl_perm_get_element (p, i);

        _gdsl_bstree_insert (&t, compare_integers, alloc_integer (&n), &rc);
        }

    _gdsl_bstree_write_xml (t, my_write_integer, stdout, "");
    _gdsl_bstree_free (t, free_integer);
    gdsl_perm_free (p);
    }

    exit (EXIT_SUCCESS);
}
```

6.9 examples/main_lllist.c

This is an example of how to use _gdsl_list module.

```
#include <config.h>


#include <stdio.h>
#include <stdlib.h>
#include <string.h>
```

```
#include "_gdsl_list.h"
#include "_strings.h"


static void
my_node_write (const _gdsl_node_t n, FILE* file, void* data)
{
    gdsl_element_t e = _gdsl_node_get_content (n);

    if (data == NULL)
    {
        fprintf (file, "%s", (char*) e);
    }
    else
    {
        fprintf (file, "%s%s", (char*) e, (char*) data);
    }
}

static int
my_node_map (const _gdsl_node_t n, void* data)
{
    my_node_write (n, stdout, data);
    return GDSL_MAP_CONT;
}

int main (void)
{
    _gdsl_list_t a = _gdsl_list_alloc (alloc_string ("a"));
    _gdsl_list_t b = _gdsl_list_alloc (alloc_string ("b"));
    _gdsl_list_t c = _gdsl_list_alloc (alloc_string ("c"));

    _gdsl_list_link (a, b);
    _gdsl_list_link (b, c);

    printf ("WRITE (%ld elements):\n", _gdsl_list_get_size (a));
    _gdsl_list_write (a, my_node_write, stdout, NULL);

    printf ("\n\nDUMP:\n");
    _gdsl_list_dump (a, my_node_write, stdout, NULL);

    printf ("\nWRITE XML:\n");
    _gdsl_list_write_xml (a, my_node_write, stdout, NULL);

    printf ("\nMAP FORWARD:\n");
    _gdsl_list_map_forward (a, my_node_map, NULL);
    printf ("\n");

    printf ("\nMAP BACKWARD:\n");
    _gdsl_list_map_backward (a, my_node_map, NULL);
    printf ("\n");

    _gdsl_list_free (a, free_string);

    exit (EXIT_SUCCESS);
}
```

6.10 examples/main_perm.c

This is an example of how to use gdsl_perm module.

```
#include <config.h>


#include <stdio.h>
#include <stdlib.h>


#include "gdsl_types.h"
#include "gdsl_perm.h"
#include "_integers.h"


static void
usage (void)
{
    printf ("Usage: perm <n>\n");
}

static void
write (const gdsl_element_t e, FILE* file, gdsl_location_t pos, void* user_data
      )
{
    ulong n = * (ulong*) e;

    if (pos & GDSL_LOCATION_FIRST)
    {
        fprintf (file, "( ");

        if (pos & GDSL_LOCATION_LAST)
        {
            fprintf (file, "%ld )\n", n);
        }
    }

    if (pos & GDSL_LOCATION_LAST)
    {
        fprintf (file, "%ld )\n", n);
    }
    else
    {
        fprintf (file, "%ld, ", n);
    }
}

int main (int argc, char* argv [])
{
    ulong i, n;
    gdsl_perm_t l_alpha;
    gdsl_perm_t c_alpha;

    if (argc < 2)
    {
        usage ();
        return EXIT_FAILURE;
    }
```

```
    n = atoi (argv[1]);

    c_alpha = gdsl_perm_alloc ("c_alpha", n);

    l_alpha = gdsl_perm_alloc ("l_alpha", n);
    gdsl_perm_randomize (l_alpha);

    printf ("alpha        = ");
    gdsl_perm_write (l_alpha, write, stdout, NULL);
    printf ("               %ld cycles, %ld inversions\n\n",
        gdsl_perm_linear_cycles_count (l_alpha),
        gdsl_perm_linear_inversions_count (l_alpha));

    gdsl_perm_reverse (l_alpha);

    printf ("~alpha       = ");
    gdsl_perm_write (l_alpha, write, stdout, NULL);
    printf ("               %ld cycles, %ld inversions\n\n",
        gdsl_perm_linear_cycles_count (l_alpha),
        gdsl_perm_linear_inversions_count (l_alpha));

    gdsl_perm_reverse (l_alpha);
    gdsl_perm_inverse (l_alpha);

    printf ("alpha^-1     = ");
    gdsl_perm_write (l_alpha, write, stdout, NULL);
    printf ("               %ld cycles, %ld inversions\n\n",
        gdsl_perm_linear_cycles_count (l_alpha),
        gdsl_perm_linear_inversions_count (l_alpha));

    gdsl_perm_inverse (l_alpha);

    printf ("alpha        = ");
    gdsl_perm_write (l_alpha, write, stdout, NULL);
    printf ("               %ld cycles, %ld inversions\n\n",
        gdsl_perm_linear_cycles_count (l_alpha),
        gdsl_perm_linear_inversions_count (l_alpha));

    gdsl_perm_linear_to_canonical (c_alpha, l_alpha);
    printf ("cycles(alpha) = ");
    gdsl_perm_write (c_alpha, write, stdout, NULL);
    printf ("               %ld cycles\n\n",
        gdsl_perm_canonical_cycles_count (c_alpha));

    gdsl_perm_canonical_to_linear (l_alpha, c_alpha);

    printf ("alpha        = ");
    gdsl_perm_write (l_alpha, write, stdout, NULL);
    printf ("               %ld cycles, %ld inversions\n\n",
        gdsl_perm_linear_cycles_count (l_alpha),
        gdsl_perm_linear_inversions_count (l_alpha));

    gdsl_perm_free (l_alpha);
    gdsl_perm_free (c_alpha);

    {
    ulong v [] = {0, 2, 3, 1, 4, 5, 6, 7, 8};
    ulong n = sizeof (v) / sizeof (v [0]);
    gdsl_perm_t a = gdsl_perm_alloc ("a", n);

    printf ("initial array: ");
    for (i = 0; i < n; i++)
```

```
        {
        printf ("%ld ", v [i]);
        }
    printf ("\n");

    gdsl_perm_randomize (a);
    printf ("applying permutation: ");
    gdsl_perm_write (a, write, stdout, NULL);
    gdsl_perm_apply_on_array ((gdsl_element_t*) v, a);
    gdsl_perm_free (a);

    printf ("modified array: ");
    for (i = 0; i < n; i++)
        {
        printf ("%ld ", v [i]);
        }
    printf ("\n");
    }

    exit (EXIT_SUCCESS);
}
```

6.11 examples/main_queue.c

This is an example of how to use gdsl_queue module.

```
#include <config.h>


#include <stdio.h>
#include <stdlib.h>
#include <string.h>


#include "gdsl_types.h"
#include "gdsl_queue.h"


#include "_integers.h"


static void
my_write_integer (gdsl_element_t e, FILE* file, gdsl_location_t location, void*
       d)
{
    int value = * (int*) e;

    if (location & GDSL_LOCATION_HEAD)
    {
        fprintf (file, "( %d", value);
    }
    else
    {
        fprintf (file, " %d", value);
    }
```

```
    if (location & GDSL_LOCATION_TAIL)
    {
        fprintf (file, " )\n");
    }
}

static int
my_display_integer (gdsl_element_t e, gdsl_location_t location, void* d)
{
    my_write_integer (e, stdout, location, d);
    return GDSL_MAP_CONT;
}

int main (void)
{
    int choice = 0;
    gdsl_queue_t q = gdsl_queue_alloc ("Q", alloc_integer, free_integer);

    do
    {
        printf ("\t\tMENU - QUEUE\n\n");
        printf ("\t1> Put\n");
        printf ("\t2> Pop\n");
        printf ("\t3> Get Head\n");
        printf ("\t4> Get Tail\n");
        printf ("\t5> Flush\n");
        printf ("\t6> Search\n");
        printf ("\t7> Display\n");
        printf ("\t8> Dump\n");
        printf ("\t9> XML display\n");
        printf ("\t0> Quit\n\n" );
        printf ("\t\tYour choice: " );
        scanf ("%d", &choice );

        switch (choice)
        {
        case 1:
            {
            int value;
            printf ("Enter an integer value: ");
            scanf ("%d", &value);
            gdsl_queue_insert (q, (void*) &value);
            }
            break;

        case 2:
            if (!gdsl_queue_is_empty (q))
            {
                int* value = (int*) gdsl_queue_remove (q);
                printf ("Value: %d\n", *value);
                free_integer (value);
            }
            else
            {
                printf ("The queue '%s' is empty\n", gdsl_queue_get_name (q));
            }
            break;

        case 3:
            {
            if (!gdsl_queue_is_empty (q))
                {
```

```
            int head = *(int*) gdsl_queue_get_head (q);
            printf ("Head = %d\n", head);
            }
        else
            {
            printf ("The queue '%s' is empty\n", gdsl_queue_get_name (q));
            }
        }
        break;

    case 4:
        {
        if (!gdsl_queue_is_empty (q))
            {
            int tail = *(int*) gdsl_queue_get_tail (q);
            printf ("Tail = %d\n", tail);
            }
        else
            {
            printf ("The queue '%s' is empty\n", gdsl_queue_get_name (q));
            }
        }
        break;

    case 5:
        if (gdsl_queue_is_empty (q))
        {
            printf ("The queue '%s' is empty\n", gdsl_queue_get_name (q));
        }
        else
        {
            gdsl_queue_flush (q);
        }
        break;

    case 6:
        {
        int pos;
        int* value;
        printf ("Enter an integer value to search an element by its
   position: ");
        scanf ("%d", &pos);

        value = (int*) gdsl_queue_search_by_position (q, pos);
        if (value != NULL)
            {
            printf ("Value found at position %d = %d\n", pos, *value);
            }
        }
        break;

    case 7:
        if (gdsl_queue_is_empty (q))
        {
            printf ("The queue '%s' is empty\n", gdsl_queue_get_name (q));
        }
        else
        {
            printf ("%s = ", gdsl_queue_get_name (q));
            gdsl_queue_map_forward (q, my_display_integer, NULL);
        }
        break;
```

```
        case 8:
            gdsl_queue_dump (q, my_write_integer, stdout, NULL);
            break;

        case 9:
            gdsl_queue_write_xml (q, my_write_integer, stdout, NULL);
            break;
        }
    }
    while (choice != 0);

    gdsl_queue_free (q);

    exit (EXIT_SUCCESS);
}
```

6.12 examples/main_rbtree.c

This is an example of how to use gdsl_rbtree module.

```
#include <config.h>


#include <stdio.h>
#include <string.h>
#include <stdlib.h>


#include "gdsl_perm.h"
#include "gdsl_types.h"
#include "gdsl_rbtree.h"
#include "_strings.h"
#include "_integers.h"


#define N 100

static int
infix_map_f (const gdsl_element_t e,
         gdsl_location_t location,
         void* user_data)
{
    printf ("%s ", (char*) e);
    if (strcmp ((char*) e, "STOP") == 0) return GDSL_MAP_STOP;
    return GDSL_MAP_CONT;
}

int main (void)
{
    int choice;
    char name[50];
    gdsl_rbtree_t t = gdsl_rbtree_alloc ("STRINGS", alloc_string, free_string,
      compare_strings);

    do
```

```
    {
        printf ("\t\tMENU - RBTREE\n\n");
        printf ("\t 1> Insert\n");
        printf ("\t 2> Remove\n");
        printf ("\t 3> Flush\n");
        printf ("\t 4> Root's content\n");
        printf ("\t 5> Size\n");
        printf ("\t 6> Height\n");
        printf ("\t 7> Search\n");
        printf ("\t 8> Display\n");
        printf ("\t 9> XML display\n");
        printf ("\t10> Dump\n");
        printf ("\t11> Insertion of a random permutation\n");
        printf ("\t12> Prefix parse (stop if 'STOP' is found as a value)\n");
        printf ("\t13> Infix parse (stop if 'STOP' is found as a value)\n");
        printf ("\t14> Postfix parse (stop if 'STOP' is found as a value)\n");
        printf ("\t 0> Quit\n\n");
        printf ("\t\tYour choice: ");
        scanf ("%d", &choice);

        switch (choice)
        {
        case 1:
            {
            int rc;

            printf ("Enter a string: ");
            scanf ("%s", name);

            gdsl_rbtree_insert (t, (void *) name, &rc);

            if (rc == GDSL_FOUND)
                {
                printf ("'%s' is already into the tree\n", name);
                }

            if (rc == GDSL_ERR_MEM_ALLOC)
                {
                printf ("memory allocation error\n");
                }
            }
            break;

        case 2:
            if (gdsl_rbtree_is_empty (t))
            {
                printf ("The tree is empty\n");
            }
            else
            {
                printf ("Enter a string: ");
                scanf ("%s", name);

                if (gdsl_rbtree_delete (t, (void*) name))
                {
                    printf ("String '%s' removed from the tree\n", name);
                }
                else
                {
                    printf ("String '%s' not found\n", name);
                }
            }
```

```
            break;

        case 3:
            gdsl_rbtree_flush (t);
            break;

        case 4:
            if (gdsl_rbtree_is_empty (t))
            {
                printf ("The tree is empty\n");
            }
            else
            {
                print_string ((char*) gdsl_rbtree_get_root (t), stdout,
      GDSL_LOCATION_ROOT, (void*) "\n");
            }
            break;

        case 5:
            printf ("Tree's size: %lu\n", gdsl_rbtree_get_size (t));
            break;

        case 6:
            printf ("Tree's height: %lu\n", gdsl_rbtree_height (t));
            break;

        case 7:
            printf( "Enter a string: " );
            scanf( "%s", name );

            if (gdsl_rbtree_search (t, NULL, (void*) name))
            {
                printf ("String '%s' found\n", name);
            }
            else
            {
                printf ("String '%s' not found\n", name);
            }
            break;

        case 8:
            if (gdsl_rbtree_is_empty (t))
            {
                printf ("The tree is empty\n");
            }
            else
            {
                printf ("Tree's content: ");
                gdsl_rbtree_write (t, print_string, stdout, (void*) " ");
                printf ("\n");
            }
            break;

        case 9:
            gdsl_rbtree_write_xml (t, print_string, stdout, NULL);
            break;

        case 10:
            gdsl_rbtree_dump (t, print_string, stdout, NULL);
            break;

        case 11:
```

```
            {
            int i;
            int rc;
            gdsl_perm_t   p = gdsl_perm_alloc ("p", N);
            gdsl_rbtree_t nt = gdsl_rbtree_alloc ("INTEGERS", alloc_integer,
      free_integer, compare_integers);

            gdsl_perm_randomize (p);

            for (i = 0; i < N; i++)
                {
                int n = gdsl_perm_get_element (p, i);
                gdsl_rbtree_insert (nt, &n, &rc);
                }

            printf ("Tree's height: %lu\n", gdsl_rbtree_height (nt));
            gdsl_rbtree_dump (nt, print_integer, stdout, "");

            gdsl_rbtree_free (nt);
            gdsl_perm_free (p);
            }
            break;

        case 12:
            gdsl_rbtree_map_prefix (t, infix_map_f, NULL);
            break;

        case 13:
            gdsl_rbtree_map_infix (t, infix_map_f, NULL);
            break;

        case 14:
            gdsl_rbtree_map_postfix (t, infix_map_f, NULL);
            break;
        }
    }
    while (choice != 0);

    gdsl_rbtree_free (t);

    exit (EXIT_SUCCESS);
}
```

6.13 examples/main_sort.c

This is an example of how to use gdsl_sort module.

```
#include <config.h>


#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/time.h>
```

```
#include "gdsl_types.h"
#include "gdsl_sort.h"


#define N 100
#define M 26


static long int
comp_char (const gdsl_element_t i, void* j)
{
    return (long int) i - (long int) j;
}


int main (void)
{
    int i;
    long int numbers [N];
    struct timeval tv;

    gettimeofday (&tv, NULL);
    srand (tv.tv_usec);

    printf ("Array of %d elements not sorted:\n", N);
    for (i = 0; i < N; i++)
    {
        numbers [i] = 'a' + (long int) ((double) M * rand() / (RAND_MAX + 1.0))
      ;
        printf ("%c ", (char) numbers [i]);
    }
    printf ("\n");

    gdsl_sort ((gdsl_element_t*) numbers, N, comp_char);

    printf ("Array sorted:\n");
    for (i = 0; i < N; i++)
    {
        printf ("%c ", (char) numbers [i]);
    }
    printf ("\n");

    exit (EXIT_SUCCESS);
}
```

6.14 examples/main_stack.c

This is an example of how to use gdsl_stack module.

```
#include <config.h>


#include <stdio.h>
#include <stdlib.h>
#include <string.h>
```

```
#include "gdsl_types.h"
#include "gdsl_stack.h"


#include "_integers.h"


static int
my_display_integer (gdsl_element_t e, gdsl_location_t location, void *
      user_infos)
{
    int* f = (int*) e;
    printf ("%d ", *f);
    return GDSL_MAP_CONT;
}

int main (void)
{
    int choix = 0;
    gdsl_stack_t s = gdsl_stack_alloc ("S", alloc_integer, free_integer);

    do
    {
        printf ("\t\tMENU - STACK\n\n");
        printf ("\t1> Push\n");
        printf ("\t2> Pop\n");
        printf ("\t3> Get\n");
        printf ("\t4> Flush\n");
        printf ("\t5> Search\n");
        printf ("\t6> Display\n");
        printf ("\t7> Dump\n");
        printf ("\t8> XML display\n");
        printf ("\t0> Quit\n\n");
        printf ("\t\tYour choice: ");
        scanf ("%d", &choix);

        switch (choix)
        {
        case 1:
            {
            int value;

            printf ("Enter integer value: ");
            scanf ("%d", &value);
            gdsl_stack_insert (s, (void*) &value);
            }
            break;

        case 2:
            if (!gdsl_stack_is_empty (s))
            {
                free_integer (gdsl_stack_remove (s));
            }
            else
            {
                printf ("The stack '%s' is empty\n", gdsl_stack_get_name (s));
            }
            break;

        case 3:
            {
```

```
            int* top;

            if (!gdsl_stack_is_empty (s))
                {
                top = (int*) gdsl_stack_get_top (s);
                printf ("Value = %d\n", *top);
                }
            else
                {
                printf ("The stack '%s' is empty\n", gdsl_stack_get_name (s));
                }
            }
            break;

        case 4:
            if (gdsl_stack_is_empty (s))
            {
                printf ("The stack '%s' is empty\n", gdsl_stack_get_name (s));
            }
            else
            {
                gdsl_stack_flush (s);
            }
            break;

        case 5:
            {
            int pos;
            int* value;
            printf ("Enter an integer value to search an element by its
       position: ");
            scanf ("%d", &pos);

            value = (int*) gdsl_stack_search_by_position (s, pos);
            if (value != NULL)
                {
                printf ("Value found at position %d = %d\n", pos, *value);
                }
            }
            break;

        case 6:
            if (gdsl_stack_is_empty (s))
            {
                printf ("The stack '%s' is empty\n", gdsl_stack_get_name (s));
            }
            else
            {
                printf ("%s = ( ", gdsl_stack_get_name (s));
                gdsl_stack_map_forward (s, my_display_integer, NULL);
                printf (")\n");
            }
            break;


        case 7:
            gdsl_stack_dump (s, print_integer, stdout, NULL);
            break;

        case 8:
            gdsl_stack_write_xml (s, print_integer, stdout, NULL);
            break;
```

```
        }
    }
    while (choix != 0);

    gdsl_stack_free (s);

    exit (EXIT_SUCCESS);
}
```

Index