

gdsI

1.6

Generated by Doxygen 1.7.6.1

Tue Aug 21 2012 16:00:01

Contents

1	gdsl	1
1.1	Introduction	1
1.2	About	1
1.2.1	Authors	1
1.2.2	Project Manager	1
2	Module Index	3
2.1	Modules	3
3	File Index	5
3.1	File List	5
4	Module Documentation	7
4.1	Low level binary tree manipulation module	7
4.1.1	Typedef Documentation	9
4.1.1.1	_gdsl_bintree_t	9
4.1.1.2	_gdsl_bintree_map_func_t	9
4.1.1.3	_gdsl_bintree_write_func_t	9
4.1.2	Function Documentation	10
4.1.2.1	_gdsl_bintree_alloc	10
4.1.2.2	_gdsl_bintree_free	10
4.1.2.3	_gdsl_bintree_copy	11
4.1.2.4	_gdsl_bintree_is_empty	12
4.1.2.5	_gdsl_bintree_is_leaf	12
4.1.2.6	_gdsl_bintree_is_root	13
4.1.2.7	_gdsl_bintree_get_content	13

4.1.2.8	<code>_gdsl_bintree_get_parent</code>	14
4.1.2.9	<code>_gdsl_bintree_get_left</code>	14
4.1.2.10	<code>_gdsl_bintree_get_right</code>	15
4.1.2.11	<code>_gdsl_bintree_get_left_ref</code>	15
4.1.2.12	<code>_gdsl_bintree_get_right_ref</code>	16
4.1.2.13	<code>_gdsl_bintree_get_height</code>	17
4.1.2.14	<code>_gdsl_bintree_get_size</code>	17
4.1.2.15	<code>_gdsl_bintree_set_content</code>	18
4.1.2.16	<code>_gdsl_bintree_set_parent</code>	18
4.1.2.17	<code>_gdsl_bintree_set_left</code>	19
4.1.2.18	<code>_gdsl_bintree_set_right</code>	19
4.1.2.19	<code>_gdsl_bintree_rotate_left</code>	20
4.1.2.20	<code>_gdsl_bintree_rotate_right</code>	20
4.1.2.21	<code>_gdsl_bintree_rotate_left_right</code>	21
4.1.2.22	<code>_gdsl_bintree_rotate_right_left</code>	21
4.1.2.23	<code>_gdsl_bintree_map_prefix</code>	22
4.1.2.24	<code>_gdsl_bintree_map_infix</code>	23
4.1.2.25	<code>_gdsl_bintree_map_postfix</code>	24
4.1.2.26	<code>_gdsl_bintree_write</code>	24
4.1.2.27	<code>_gdsl_bintree_write_xml</code>	25
4.1.2.28	<code>_gdsl_bintree_dump</code>	26
4.2	Low-level binary search tree manipulation module	27
4.2.1	Typedef Documentation	28
4.2.1.1	<code>_gdsl_bstree_t</code>	28
4.2.1.2	<code>_gdsl_bstree_map_func_t</code>	28
4.2.1.3	<code>_gdsl_bstree_write_func_t</code>	29
4.2.2	Function Documentation	29
4.2.2.1	<code>_gdsl_bstree_alloc</code>	29
4.2.2.2	<code>_gdsl_bstree_free</code>	30
4.2.2.3	<code>_gdsl_bstree_copy</code>	30
4.2.2.4	<code>_gdsl_bstree_is_empty</code>	31
4.2.2.5	<code>_gdsl_bstree_is_leaf</code>	32
4.2.2.6	<code>_gdsl_bstree_get_content</code>	32
4.2.2.7	<code>_gdsl_bstree_is_root</code>	33

4.2.2.8	_gdsl_bstree_get_parent	33
4.2.2.9	_gdsl_bstree_get_left	34
4.2.2.10	_gdsl_bstree_get_right	34
4.2.2.11	_gdsl_bstree_get_size	35
4.2.2.12	_gdsl_bstree_get_height	35
4.2.2.13	_gdsl_bstree_insert	36
4.2.2.14	_gdsl_bstree_remove	37
4.2.2.15	_gdsl_bstree_search	37
4.2.2.16	_gdsl_bstree_search_next	38
4.2.2.17	_gdsl_bstree_map_prefix	39
4.2.2.18	_gdsl_bstree_map_infix	39
4.2.2.19	_gdsl_bstree_map_postfix	40
4.2.2.20	_gdsl_bstree_write	41
4.2.2.21	_gdsl_bstree_write_xml	41
4.2.2.22	_gdsl_bstree_dump	42
4.3	Low-level doubly-linked list manipulation module	44
4.3.1	Typedef Documentation	45
4.3.1.1	_gdsl_list_t	45
4.3.2	Function Documentation	45
4.3.2.1	_gdsl_list_alloc	45
4.3.2.2	_gdsl_list_free	45
4.3.2.3	_gdsl_list_is_empty	46
4.3.2.4	_gdsl_list_get_size	46
4.3.2.5	_gdsl_list_link	47
4.3.2.6	_gdsl_list_insert_after	47
4.3.2.7	_gdsl_list_insert_before	48
4.3.2.8	_gdsl_list_remove	48
4.3.2.9	_gdsl_list_search	49
4.3.2.10	_gdsl_list_map_forward	49
4.3.2.11	_gdsl_list_map_backward	50
4.3.2.12	_gdsl_list_write	51
4.3.2.13	_gdsl_list_write_xml	51
4.3.2.14	_gdsl_list_dump	52
4.4	Low-level doubly-linked node manipulation module	53

4.4.1	Typedef Documentation	54
4.4.1.1	_gdsl_node_t	54
4.4.1.2	_gdsl_node_map_func_t	54
4.4.1.3	_gdsl_node_write_func_t	54
4.4.2	Function Documentation	55
4.4.2.1	_gdsl_node_alloc	55
4.4.2.2	_gdsl_node_free	55
4.4.2.3	_gdsl_node_get_succ	56
4.4.2.4	_gdsl_node_get_pred	56
4.4.2.5	_gdsl_node_get_content	57
4.4.2.6	_gdsl_node_set_succ	57
4.4.2.7	_gdsl_node_set_pred	58
4.4.2.8	_gdsl_node_set_content	58
4.4.2.9	_gdsl_node_link	59
4.4.2.10	_gdsl_node_unlink	59
4.4.2.11	_gdsl_node_write	60
4.4.2.12	_gdsl_node_write_xml	60
4.4.2.13	_gdsl_node_dump	61
4.5	Main module	63
4.5.1	Function Documentation	63
4.5.1.1	gdsl_get_version	63
4.6	2D-Arrays manipulation module	64
4.6.1	Typedef Documentation	65
4.6.1.1	gdsl_2darray_t	65
4.6.2	Function Documentation	65
4.6.2.1	gdsl_2darray_alloc	65
4.6.2.2	gdsl_2darray_free	66
4.6.2.3	gdsl_2darray_get_name	66
4.6.2.4	gdsl_2darray_get_rows_number	67
4.6.2.5	gdsl_2darray_get_columns_number	67
4.6.2.6	gdsl_2darray_get_size	68
4.6.2.7	gdsl_2darray_get_content	68
4.6.2.8	gdsl_2darray_set_name	69
4.6.2.9	gdsl_2darray_set_content	70

4.6.2.10	gdsl_2darray_write	70
4.6.2.11	gdsl_2darray_write_xml	71
4.6.2.12	gdsl_2darray_dump	72
4.7	Binary search tree manipulation module	73
4.7.1	Typedef Documentation	74
4.7.1.1	gdsl_bstree_t	74
4.7.2	Function Documentation	74
4.7.2.1	gdsl_bstree_alloc	74
4.7.2.2	gdsl_bstree_free	75
4.7.2.3	gdsl_bstree_flush	76
4.7.2.4	gdsl_bstree_get_name	76
4.7.2.5	gdsl_bstree_is_empty	77
4.7.2.6	gdsl_bstree_get_root	77
4.7.2.7	gdsl_bstree_get_size	78
4.7.2.8	gdsl_bstree_get_height	78
4.7.2.9	gdsl_bstree_set_name	79
4.7.2.10	gdsl_bstree_insert	79
4.7.2.11	gdsl_bstree_remove	80
4.7.2.12	gdsl_bstree_delete	81
4.7.2.13	gdsl_bstree_search	81
4.7.2.14	gdsl_bstree_map_prefix	82
4.7.2.15	gdsl_bstree_map_infix	83
4.7.2.16	gdsl_bstree_map_postfix	84
4.7.2.17	gdsl_bstree_write	84
4.7.2.18	gdsl_bstree_write_xml	85
4.7.2.19	gdsl_bstree_dump	86
4.8	Hashtable manipulation module	87
4.8.1	Typedef Documentation	88
4.8.1.1	gdsl_hash_t	88
4.8.1.2	gdsl_key_func_t	88
4.8.1.3	gdsl_hash_func_t	89
4.8.2	Function Documentation	89
4.8.2.1	gdsl_hash	89
4.8.2.2	gdsl_hash_alloc	89

4.8.2.3	gdsl_hash_free	90
4.8.2.4	gdsl_hash_flush	91
4.8.2.5	gdsl_hash_get_name	92
4.8.2.6	gdsl_hash_get_entries_number	92
4.8.2.7	gdsl_hash_get_lists_max_size	93
4.8.2.8	gdsl_hash_get_longest_list_size	93
4.8.2.9	gdsl_hash_get_size	94
4.8.2.10	gdsl_hash_get_fill_factor	95
4.8.2.11	gdsl_hash_set_name	95
4.8.2.12	gdsl_hash_insert	96
4.8.2.13	gdsl_hash_remove	97
4.8.2.14	gdsl_hash_delete	97
4.8.2.15	gdsl_hash_modify	98
4.8.2.16	gdsl_hash_search	99
4.8.2.17	gdsl_hash_map	100
4.8.2.18	gdsl_hash_write	100
4.8.2.19	gdsl_hash_write_xml	101
4.8.2.20	gdsl_hash_dump	102
4.9	Heap manipulation module	103
4.9.1	Typedef Documentation	104
4.9.1.1	gdsl_heap_t	104
4.9.2	Function Documentation	104
4.9.2.1	gdsl_heap_alloc	104
4.9.2.2	gdsl_heap_free	105
4.9.2.3	gdsl_heap_flush	105
4.9.2.4	gdsl_heap_get_name	106
4.9.2.5	gdsl_heap_get_size	106
4.9.2.6	gdsl_heap_get_top	107
4.9.2.7	gdsl_heap_is_empty	107
4.9.2.8	gdsl_heap_set_name	108
4.9.2.9	gdsl_heap_set_top	109
4.9.2.10	gdsl_heap_insert	109
4.9.2.11	gdsl_heap_remove_top	110
4.9.2.12	gdsl_heap_delete_top	110

4.9.2.13	gdsl_heap_map_forward	111
4.9.2.14	gdsl_heap_write	112
4.9.2.15	gdsl_heap_write_xml	112
4.9.2.16	gdsl_heap_dump	113
4.10	Interval Heap manipulation module	114
4.10.1	Typedef Documentation	115
4.10.1.1	gdsl_interval_heap_t	115
4.10.2	Function Documentation	115
4.10.2.1	gdsl_interval_heap_alloc	115
4.10.2.2	gdsl_interval_heap_free	116
4.10.2.3	gdsl_interval_heap_flush	117
4.10.2.4	gdsl_interval_heap_get_name	117
4.10.2.5	gdsl_interval_heap_get_size	118
4.10.2.6	gdsl_interval_heap_set_max_size	118
4.10.2.7	gdsl_interval_heap_is_empty	119
4.10.2.8	gdsl_interval_heap_set_name	119
4.10.2.9	gdsl_interval_heap_insert	120
4.10.2.10	gdsl_interval_heap_remove_max	121
4.10.2.11	gdsl_interval_heap_remove_min	121
4.10.2.12	gdsl_interval_heap_get_min	122
4.10.2.13	gdsl_interval_heap_get_max	122
4.10.2.14	gdsl_interval_heap_delete_min	123
4.10.2.15	gdsl_interval_heap_delete_max	123
4.10.2.16	gdsl_interval_heap_map_forward	124
4.10.2.17	gdsl_interval_heap_write	125
4.10.2.18	gdsl_interval_heap_write_xml	125
4.10.2.19	gdsl_interval_heap_dump	126
4.11	Doubly-linked list manipulation module	127
4.11.1	Typedef Documentation	130
4.11.1.1	gdsl_list_t	130
4.11.1.2	gdsl_list_cursor_t	130
4.11.2	Function Documentation	130
4.11.2.1	gdsl_list_alloc	130
4.11.2.2	gdsl_list_free	131

4.11.2.3	gdsl_list_flush	131
4.11.2.4	gdsl_list_get_name	132
4.11.2.5	gdsl_list_get_size	132
4.11.2.6	gdsl_list_is_empty	133
4.11.2.7	gdsl_list_get_head	133
4.11.2.8	gdsl_list_get_tail	134
4.11.2.9	gdsl_list_set_name	134
4.11.2.10	gdsl_list_insert_head	135
4.11.2.11	gdsl_list_insert_tail	136
4.11.2.12	gdsl_list_remove_head	136
4.11.2.13	gdsl_list_remove_tail	137
4.11.2.14	gdsl_list_remove	138
4.11.2.15	gdsl_list_delete_head	139
4.11.2.16	gdsl_list_delete_tail	139
4.11.2.17	gdsl_list_delete	140
4.11.2.18	gdsl_list_search	141
4.11.2.19	gdsl_list_search_by_position	141
4.11.2.20	gdsl_list_search_max	142
4.11.2.21	gdsl_list_search_min	143
4.11.2.22	gdsl_list_sort	143
4.11.2.23	gdsl_list_map_forward	144
4.11.2.24	gdsl_list_map_backward	144
4.11.2.25	gdsl_list_write	145
4.11.2.26	gdsl_list_write_xml	146
4.11.2.27	gdsl_list_dump	147
4.11.2.28	gdsl_list_cursor_alloc	147
4.11.2.29	gdsl_list_cursor_free	148
4.11.2.30	gdsl_list_cursor_move_to_head	148
4.11.2.31	gdsl_list_cursor_move_to_tail	149
4.11.2.32	gdsl_list_cursor_move_to_value	149
4.11.2.33	gdsl_list_cursor_move_to_position	150
4.11.2.34	gdsl_list_cursor_step_forward	150
4.11.2.35	gdsl_list_cursor_step_backward	151
4.11.2.36	gdsl_list_cursor_is_on_head	151

4.11.2.37	gdsl_list_cursor_is_on_tail	152
4.11.2.38	gdsl_list_cursor_has_succ	152
4.11.2.39	gdsl_list_cursor_has_pred	153
4.11.2.40	gdsl_list_cursor_set_content	154
4.11.2.41	gdsl_list_cursor_get_content	154
4.11.2.42	gdsl_list_cursor_insert_after	155
4.11.2.43	gdsl_list_cursor_insert_before	155
4.11.2.44	gdsl_list_cursor_remove	156
4.11.2.45	gdsl_list_cursor_remove_after	157
4.11.2.46	gdsl_list_cursor_remove_before	157
4.11.2.47	gdsl_list_cursor_delete	158
4.11.2.48	gdsl_list_cursor_delete_after	159
4.11.2.49	gdsl_list_cursor_delete_before	159
4.12	Various macros module	161
4.12.1	Define Documentation	161
4.12.1.1	GDSL_MAX	161
4.12.1.2	GDSL_MIN	161
4.13	Permutation manipulation module	163
4.13.1	Typedef Documentation	164
4.13.1.1	gdsl_perm_t	164
4.13.1.2	gdsl_perm_write_func_t	165
4.13.1.3	gdsl_perm_data_t	165
4.13.2	Enumeration Type Documentation	165
4.13.2.1	gdsl_perm_position_t	165
4.13.3	Function Documentation	165
4.13.3.1	gdsl_perm_alloc	165
4.13.3.2	gdsl_perm_free	166
4.13.3.3	gdsl_perm_copy	166
4.13.3.4	gdsl_perm_get_name	167
4.13.3.5	gdsl_perm_get_size	168
4.13.3.6	gdsl_perm_get_element	168
4.13.3.7	gdsl_perm_get_elements_array	169
4.13.3.8	gdsl_perm_linear_inversions_count	169
4.13.3.9	gdsl_perm_linear_cycles_count	170

4.13.3.10	gdsl_perm_canonical_cycles_count	170
4.13.3.11	gdsl_perm_set_name	171
4.13.3.12	gdsl_perm_linear_next	172
4.13.3.13	gdsl_perm_linear_prev	172
4.13.3.14	gdsl_perm_set_elements_array	173
4.13.3.15	gdsl_perm_multiply	173
4.13.3.16	gdsl_perm_linear_to_canonical	174
4.13.3.17	gdsl_perm_canonical_to_linear	175
4.13.3.18	gdsl_perm_inverse	175
4.13.3.19	gdsl_perm_reverse	176
4.13.3.20	gdsl_perm_randomize	176
4.13.3.21	gdsl_perm_apply_on_array	177
4.13.3.22	gdsl_perm_write	177
4.13.3.23	gdsl_perm_write_xml	178
4.13.3.24	gdsl_perm_dump	179
4.14	Queue manipulation module	180
4.14.1	Typedef Documentation	181
4.14.1.1	gdsl_queue_t	181
4.14.2	Function Documentation	181
4.14.2.1	gdsl_queue_alloc	181
4.14.2.2	gdsl_queue_free	182
4.14.2.3	gdsl_queue_flush	182
4.14.2.4	gdsl_queue_get_name	183
4.14.2.5	gdsl_queue_get_size	183
4.14.2.6	gdsl_queue_is_empty	184
4.14.2.7	gdsl_queue_get_head	184
4.14.2.8	gdsl_queue_get_tail	185
4.14.2.9	gdsl_queue_set_name	186
4.14.2.10	gdsl_queue_insert	186
4.14.2.11	gdsl_queue_remove	187
4.14.2.12	gdsl_queue_search	187
4.14.2.13	gdsl_queue_search_by_position	188
4.14.2.14	gdsl_queue_map_forward	189
4.14.2.15	gdsl_queue_map_backward	189

4.14.2.16	gdsl_queue_write	190
4.14.2.17	gdsl_queue_write_xml	191
4.14.2.18	gdsl_queue_dump	191
4.15	Red-black tree manipulation module	193
4.15.1	Typedef Documentation	194
4.15.1.1	gdsl_rbtrees_t	194
4.15.2	Function Documentation	194
4.15.2.1	gdsl_rbtrees_alloc	194
4.15.2.2	gdsl_rbtrees_free	195
4.15.2.3	gdsl_rbtrees_flush	196
4.15.2.4	gdsl_rbtrees_get_name	196
4.15.2.5	gdsl_rbtrees_is_empty	197
4.15.2.6	gdsl_rbtrees_get_root	197
4.15.2.7	gdsl_rbtrees_get_size	198
4.15.2.8	gdsl_rbtrees_height	198
4.15.2.9	gdsl_rbtrees_set_name	199
4.15.2.10	gdsl_rbtrees_insert	199
4.15.2.11	gdsl_rbtrees_remove	200
4.15.2.12	gdsl_rbtrees_delete	201
4.15.2.13	gdsl_rbtrees_search	201
4.15.2.14	gdsl_rbtrees_map_prefix	202
4.15.2.15	gdsl_rbtrees_map_infix	203
4.15.2.16	gdsl_rbtrees_map_postfix	204
4.15.2.17	gdsl_rbtrees_write	204
4.15.2.18	gdsl_rbtrees_write_xml	205
4.15.2.19	gdsl_rbtrees_dump	206
4.16	Sort module	207
4.16.1	Function Documentation	207
4.16.1.1	gdsl_sort	207
4.17	Stack manipulation module	208
4.17.1	Typedef Documentation	209
4.17.1.1	gdsl_stack_t	209
4.17.2	Function Documentation	209
4.17.2.1	gdsl_stack_alloc	209

4.17.2.2	gdsl_stack_free	210
4.17.2.3	gdsl_stack_flush	211
4.17.2.4	gdsl_stack_get_name	211
4.17.2.5	gdsl_stack_get_size	212
4.17.2.6	gdsl_stack_get_growing_factor	212
4.17.2.7	gdsl_stack_is_empty	213
4.17.2.8	gdsl_stack_get_top	213
4.17.2.9	gdsl_stack_get_bottom	214
4.17.2.10	gdsl_stack_set_name	214
4.17.2.11	gdsl_stack_set_growing_factor	215
4.17.2.12	gdsl_stack_insert	216
4.17.2.13	gdsl_stack_remove	216
4.17.2.14	gdsl_stack_search	217
4.17.2.15	gdsl_stack_search_by_position	218
4.17.2.16	gdsl_stack_map_forward	218
4.17.2.17	gdsl_stack_map_backward	219
4.17.2.18	gdsl_stack_write	220
4.17.2.19	gdsl_stack_write_xml	220
4.17.2.20	gdsl_stack_dump	221
4.18	GDSL types	222
4.18.1	Typedef Documentation	222
4.18.1.1	gdsl_element_t	222
4.18.1.2	gdsl_alloc_func_t	223
4.18.1.3	gdsl_free_func_t	223
4.18.1.4	gdsl_copy_func_t	223
4.18.1.5	gdsl_map_func_t	224
4.18.1.6	gdsl_compare_func_t	224
4.18.1.7	gdsl_write_func_t	225
4.18.1.8	ulong	225
4.18.1.9	ushort	225
4.18.2	Enumeration Type Documentation	225
4.18.2.1	gdsl_constant_t	225
4.18.2.2	gdsl_location_t	226
4.18.2.3	bool	226

5	File Documentation	227
5.1	_gdsl_bintree.h File Reference	227
5.2	_gdsl_bstree.h File Reference	229
5.3	_gdsl_list.h File Reference	230
5.4	_gdsl_node.h File Reference	231
5.5	gdsl.h File Reference	233
5.6	gdsl_2darray.h File Reference	233
5.7	gdsl_bstree.h File Reference	234
5.8	gdsl_hash.h File Reference	235
5.9	gdsl_heap.h File Reference	237
5.10	gdsl_interval_heap.h File Reference	238
5.11	gdsl_list.h File Reference	239
5.12	gdsl_macros.h File Reference	242
5.13	gdsl_perm.h File Reference	242
5.14	gdsl_queue.h File Reference	244
5.15	gdsl_rbtrees.h File Reference	245
5.16	gdsl_sort.h File Reference	247
5.17	gdsl_stack.h File Reference	247
5.18	gdsl_types.h File Reference	248
5.19	mainpage.h File Reference	249

Chapter 1

gdsl

1.1 Introduction

This is the gdsl (Release 1.6) documentation.

1.2 About

The Generic Data Structures Library (GDSDL) 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, binary search trees, red-black trees, 2D arrays, permutations and heaps.

1.2.1 Authors

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

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

1.2.2 Project Manager

Nicolas Darnis <ndarnis@free.fr>

Chapter 2

Module Index

2.1 Modules

Here is a list of all modules:

Low level binary tree manipulation module	7
Low-level binary search tree manipulation module	27
Low-level doubly-linked list manipulation module	44
Low-level doubly-linked node manipulation module	53
Main module	63
2D-Arrays manipulation module	64
Binary search tree manipulation module	73
Hashtable manipulation module	87
Heap manipulation module	103
Interval Heap manipulation module	114
Doubly-linked list manipulation module	127
Various macros module	161
Permutation manipulation module	163
Queue manipulation module	180
Red-black tree manipulation module	193
Sort module	207
Stack manipulation module	208
GDSDL types	222

Chapter 3

File Index

3.1 File List

Here is a list of all files with brief descriptions:

<code>_gdsi_bintree.h</code>	227
<code>_gdsi_bstree.h</code>	229
<code>_gdsi_list.h</code>	230
<code>_gdsi_node.h</code>	231
<code>gdsi.h</code>	233
<code>gdsi_2darray.h</code>	233
<code>gdsi_bstree.h</code>	234
<code>gdsi_hash.h</code>	235
<code>gdsi_heap.h</code>	237
<code>gdsi_interval_heap.h</code>	238
<code>gdsi_list.h</code>	239
<code>gdsi_macros.h</code>	242
<code>gdsi_perm.h</code>	242
<code>gdsi_queue.h</code>	244
<code>gdsi_rbtrees.h</code>	245
<code>gdsi_sort.h</code>	247
<code>gdsi_stack.h</code>	247
<code>gdsi_types.h</code>	248
<code>mainpage.h</code>	249

Chapter 4

Module Documentation

4.1 Low level binary tree manipulation module

Typedefs

- `typedef struct _gdsI_bintree * _gdsI_bintree_t`
GDSL low-level binary tree type.
- `typedef int(* _gdsI_bintree_map_func_t)(const _gdsI_bintree_t TREE, void *USER_DATA)`
GDSL low-level binary tree map function type.
- `typedef void(* _gdsI_bintree_write_func_t)(const _gdsI_bintree_t TREE, FILE *OUTPUT_FILE, void *USER_DATA)`
GDSL low-level binary tree write function type.

Functions

- `_gdsI_bintree_t _gdsI_bintree_alloc` (const `gdsI_element_t` E, const `_gdsI_bintree_t` LEFT, const `_gdsI_bintree_t` RIGHT)
Create a new low-level binary tree.
- `void _gdsI_bintree_free` (`_gdsI_bintree_t` T, const `gdsI_free_func_t` FREE_F)
Destroy a low-level binary tree.
- `_gdsI_bintree_t _gdsI_bintree_copy` (const `_gdsI_bintree_t` T, const `gdsI_copy_func_t` COPY_F)
Copy a low-level binary tree.
- `bool _gdsI_bintree_is_empty` (const `_gdsI_bintree_t` T)
Check if a low-level binary tree is empty.
- `bool _gdsI_bintree_is_leaf` (const `_gdsI_bintree_t` T)
Check if a low-level binary tree is reduced to a leaf.
- `bool _gdsI_bintree_is_root` (const `_gdsI_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 infix 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 postfix order.

- void **_gdsI_bintree_write** (const **_gdsI_bintree_t** T, const **_gdsI_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 **_gdsI_bintree_write_xml** (const **_gdsI_bintree_t** T, const **_gdsI_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 **_gdsI_bintree_dump** (const **_gdsI_bintree_t** T, const **_gdsI_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 Typedef Documentation

4.1.1.1 typedef struct **_gdsI_bintree*** **_gdsI_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 54 of file **_gdsI_bintree.h**.

4.1.1.2 typedef int(* **_gdsI_bintree_map_func_t**)(const **_gdsI_bintree_t** TREE, void *USER_DATA)

GDSL low-level binary tree map function type.

Parameters

<i>TREE</i>	The low-level binary tree to map.
<i>USER_DATA</i>	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 63 of file **_gdsI_bintree.h**.

4.1.1.3 typedef void(* **_gdsI_bintree_write_func_t**)(const **_gdsI_bintree_t** TREE, FILE *OUTPUT_FILE, void *USER_DATA)

GDSL low-level binary tree write function type.

Parameters

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

Definition at line 73 of file `_gdsl_bintree.h`.

4.1.2 Function Documentation

4.1.2.1 `_gdsl_bintree_t _gdsl_bintree_alloc(const gdsl_element_t E, const _gdsl_bintree_t LEFT, const _gdsl_bintree_t RIGHT)`

Create a new low-level binary tree.

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

Note

Complexity: $O(1)$

Precondition

nothing.

Parameters

<i>E</i>	The root content of the new low-level binary tree to create.
<i>LEFT</i>	The left sub-tree of the new low-level binary tree to create.
<i>RIGHT</i>	The right sub-tree of the new low-level binary tree to create.

Returns

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

See also

`_gdsl_bintree_free()` (p. 10)

4.1.2.2 `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` \neq 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

<i>T</i>	The low-level binary tree to destroy.
<i>FREE_F</i>	The function used to deallocate T's nodes contents.

See also

[`_gdsI_bintree_alloc\(\)`](#) (p. 10)

4.1.2.3 `_gdsI_bintree_t _gdsI_bintree_copy(const _gdsI_bintree_t T, const gdsI_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

<i>T</i>	The low-level binary tree to copy.
<i>COPY_F</i>	The function used to copy T's nodes contents.

Returns

a copy of T in case of success.

NULL if `_gdsI_bintree_is_empty(T) == TRUE` or in case of insufficient memory.

See also

[`_gdsI_bintree_alloc\(\)`](#) (p. 10)

[`_gdsI_bintree_free\(\)`](#) (p. 10)

[`_gdsI_bintree_is_empty\(\)`](#) (p. 12)

4.1.2.4 `bool _gdsI_bintree_is_empty(const _gdsI_bintree_t T)`

Check if a low-level binary tree is empty.

Note

Complexity: $O(1)$

Precondition

nothing.

Parameters

<code>T</code>	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

`_gdsI_bintree_is_leaf()` (p. 12)

`_gdsI_bintree_is_root()` (p. 13)

4.1.2.5 `bool _gdsI_bintree_is_leaf(const _gdsI_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 `_gdsI_bintree_t`.

Parameters

<code>T</code>	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

`_gdsI_bintree_is_empty()` (p. 12)

`_gdsI_bintree_is_root()` (p. 13)

4.1.2.6 `bool _gdsI_bintree_is_root(const _gdsI_bintree_t T)`

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

Note

Complexity: $O(1)$

Precondition

T must be a non-empty `_gdsI_bintree_t`.

Parameters

<code>T</code>	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

`_gdsI_bintree_is_empty()` (p. 12)

`_gdsI_bintree_is_leaf()` (p. 12)

4.1.2.7 `_gdsI_element_t _gdsI_bintree_get_content(const _gdsI_bintree_t T)`

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

Note

Complexity: $O(1)$

Precondition

T must be a non-empty `_gdsI_bintree_t`.

Parameters

<code>T</code>	The low-level binary tree to use.
----------------	-----------------------------------

Returns

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

See also

`_gdsI_bintree_set_content()` (p. 18)

4.1.2.8 `_gdsI_bintree_t _gdsI_bintree_get_parent(const _gdsI_bintree_t T)`

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

Note

Complexity: $O(1)$

Precondition

T must be a non-empty `_gdsI_bintree_t`.

Parameters

<code>T</code>	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

`_gdsI_bintree_is_root()` (p. 13)
`_gdsI_bintree_set_parent()` (p. 18)

4.1.2.9 `_gdsI_bintree_t _gdsI_bintree_get_left(const _gdsI_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 `_gdsI_bintree_t`.

Parameters

<code>T</code>	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. 15)
`_gdsl_bintree_set_left()` (p. 19)
`_gdsl_bintree_set_right()` (p. 19)

4.1.2.10 `_gsdl_bintree_t _gsdl_bintree_get_right(const _gsdl_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 `_gsdl_bintree_t`

Parameters

<code>T</code>	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

`_gsdl_bintree_get_left()` (p. 14)
`_gsdl_bintree_set_left()` (p. 19)
`_gsdl_bintree_set_right()` (p. 19)

4.1.2.11 `_gsdl_bintree_t* _gsdl_bintree_get_left_ref(const _gsdl_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

<code>T</code>	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. 16)

4.1.2.12 `_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

<code>T</code>	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. 15)

4.1.2.13 `ulong _gdsi_bintree_get_height(const _gdsi_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

`_gdsi_bintree_get_size()` (p. 17)

4.1.2.14 `ulong _gdsi_bintree_get_size(const _gdsi_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

`_gdsI_bintree_get_height()` (p. 17)

4.1.2.15 `void _gdsI_bintree_set_content(_gdsI_bintree_t T, const gdsI_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 `_gdsI_bintree_t`.

Parameters

<i>T</i>	The low-level binary tree to modify.
<i>E</i>	The new T's root content.

See also

`_gdsI_bintree_get_content` (p. 13)

4.1.2.16 `void _gdsI_bintree_set_parent(_gdsI_bintree_t T, const _gdsI_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 `_gdsI_bintree_t`.

Parameters

<i>T</i>	The low-level binary tree to modify.
<i>P</i>	The new T's parent.

See also

`_gdsI_bintree_get_parent()` (p. 14)

4.1.2.17 `void _gdsI_bintree_set_left(_gdsI_bintree_t T, const _gdsI_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 `_gdsI_bintree_t`.

Parameters

<i>T</i>	The low-level binary tree to modify.
<i>L</i>	The new T's left sub-tree.

See also

`_gdsI_bintree_set_right()` (p. 19)

`_gdsI_bintree_get_left()` (p. 14)

`_gdsI_bintree_get_right()` (p. 15)

4.1.2.18 `void _gdsI_bintree_set_right(_gdsI_bintree_t T, const _gdsI_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 `_gdsI_bintree_t`.

Parameters

<i>T</i>	The low-level binary tree to modify.
<i>R</i>	The new T's right sub-tree.

See also

`_gdsi_bintree_set_left()` (p. 19)
`_gdsi_bintree_get_left()` (p. 14)
`_gdsi_bintree_get_right()` (p. 15)

4.1.2.19 `_gdsi_bintree_t _gdsi_bintree_rotate_left(_gdsi_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 `_gdsi_bintree_t`.

Parameters

<code>T</code>	The low-level binary tree to rotate.
----------------	--------------------------------------

Returns

the modified T left-rotated.

See also

`_gdsi_bintree_rotate_right()` (p. 20)
`_gdsi_bintree_rotate_left_right()` (p. 21)
`_gdsi_bintree_rotate_right_left()` (p. 21)

4.1.2.20 `_gdsi_bintree_t _gdsi_bintree_rotate_right(_gdsi_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 `_gdsi_bintree_t`.

Parameters

<code>T</code>	The low-level binary tree to rotate.
----------------	--------------------------------------

Returns

the modified T right-rotated.

See also

`_gdsI_bintree_rotate_left()` (p. 20)

`_gdsI_bintree_rotate_left_right()` (p. 21)

`_gdsI_bintree_rotate_right_left()` (p. 21)

4.1.2.21 `_gdsI_bintree_t _gdsI_bintree_rotate_left_right(_gdsI_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

$T \neq \text{null}$ & $l(T) \neq \text{null}$ must be non-empty `_gdsI_bintree_t`.

Parameters

<code>T</code>	The low-level binary tree to rotate.
----------------	--------------------------------------

Returns

the modified T left-right-rotated.

See also

`_gdsI_bintree_rotate_left()` (p. 20)

`_gdsI_bintree_rotate_right()` (p. 20)

`_gdsI_bintree_rotate_right_left()` (p. 21)

4.1.2.22 `_gdsI_bintree_t _gdsI_bintree_rotate_right_left(_gdsI_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

<code>T</code>	The low-level binary tree to rotate.
----------------	--------------------------------------

Returns

the modified T right-left-rotated.

See also

`_gdsl_bintree_rotate_left()` (p. 20)
`_gdsl_bintree_rotate_right()` (p. 20)
`_gdsl_bintree_rotate_left_right()` (p. 21)

4.1.2.23 `_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. 22) stops and returns its last examined node.

Note

Complexity: $O(|T|)$

Precondition

`MAP_F` \neq `NULL`

Parameters

<code>T</code>	The low-level binary tree to map.
<code>MAP_F</code>	The map function.
<code>USER_DATA</code>	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. 23)
`_gdsl_bintree_map_postfix()` (p. 24)

4.1.2.24 `_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 infix order.

Parse all nodes of the low-level binary tree T in infix 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. 23) stops and returns its last examined node.

Note

Complexity: $O(|T|)$

Precondition

MAP_F != NULL

Parameters

<i>T</i>	The low-level binary tree to map.
<i>MAP_F</i>	The map function.
<i>USER_DATA</i>	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. 22)
`_gdsl_bintree_map_postfix()` (p. 24)

4.1.2.25 `_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 postfix order.

Parse all nodes of the low-level binary tree T in postfix 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. 24) stops and returns its last examined node.

Note

Complexity: $O(|T|)$

Precondition

MAP_F != NULL

Parameters

<i>T</i>	The low-level binary tree to map.
<i>MAP_F</i>	The map function.
<i>USER_DATA</i>	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. 22)
`_gdsl_bintree_map_infix()` (p. 23)

4.1.2.26 `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

<i>T</i>	The low-level binary tree to write.
<i>WRITE_F</i>	The write function.
<i>OUTPUT_FILE</i>	The file where to write T's nodes.
<i>USER_DATA</i>	User's datas passed to WRITE_F.

See also

`_gdsi_bintree_write_xml()` (p. 25)

`_gdsi_bintree_dump()` (p. 26)

4.1.2.27 `void _gdsi_bintree_write_xml(const _gdsi_bintree_t T, const
_gdsi_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

<i>T</i>	The low-level binary tree to write.
<i>WRITE_F</i>	The write function.
<i>OUTPUT_FILE</i>	The file where to write T's nodes.
<i>USER_DATA</i>	User's datas passed to WRITE_F.

See also

`_gdsi_bintree_write()` (p. 24)

`_gdsi_bintree_dump()` (p. 26)

4.1.2.28 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

<i>T</i>	The low-level binary tree to dump.
<i>WRITE_F</i>	The write function.
<i>OUTPUT_FILE</i>	The file where to write <i>T</i> 's nodes.
<i>USER_DATA</i>	User's datas passed to <i>WRITE_F</i> .

See also

`_gdsl_bintree_write()` (p. 24)
`_gdsl_bintree_write_xml()` (p. 25)

4.2 Low-level binary search tree manipulation module

Typedefs

- typedef **_gdsi_bintree_t _gdsi_bstree_t**
GDSL low-level binary search tree type.
- typedef int(* **_gdsi_bstree_map_func_t**)(_gdsi_bstree_t TREE, void *USER_DATA)
GDSL low-level binary search tree map function type.
- typedef void(* **_gdsi_bstree_write_func_t**)(_gdsi_bstree_t TREE, FILE *OUTPUT_FILE, void *USER_DATA)
GDSL low-level binary search tree write function type.

Functions

- **_gdsi_bstree_t _gdsi_bstree_alloc**(const **gdsi_element_t** E)
Create a new low-level binary search tree.
- void **_gdsi_bstree_free**(**_gdsi_bstree_t** T, const **gdsi_free_func_t** FREE_F)
Destroy a low-level binary search tree.
- **_gdsi_bstree_t _gdsi_bstree_copy**(const **_gdsi_bstree_t** T, const **gdsi_copy_func_t** COPY_F)
Copy a low-level binary search tree.
- **bool _gdsi_bstree_is_empty**(const **_gdsi_bstree_t** T)
Check if a low-level binary search tree is empty.
- **bool _gdsi_bstree_is_leaf**(const **_gdsi_bstree_t** T)
Check if a low-level binary search tree is reduced to a leaf.
- **gdsi_element_t _gdsi_bstree_get_content**(const **_gdsi_bstree_t** T)
Get the root content of a low-level binary search tree.
- **bool _gdsi_bstree_is_root**(const **_gdsi_bstree_t** T)
Check if a low-level binary search tree is a root.
- **_gdsi_bstree_t _gdsi_bstree_get_parent**(const **_gdsi_bstree_t** T)
Get the parent tree of a low-level binary search tree.
- **_gdsi_bstree_t _gdsi_bstree_get_left**(const **_gdsi_bstree_t** T)
Get the left sub-tree of a low-level binary search tree.
- **_gdsi_bstree_t _gdsi_bstree_get_right**(const **_gdsi_bstree_t** T)
Get the right sub-tree of a low-level binary search tree.
- **ulong _gdsi_bstree_get_size**(const **_gdsi_bstree_t** T)
Get the size of a low-level binary search tree.
- **ulong _gdsi_bstree_get_height**(const **_gdsi_bstree_t** T)
Get the height of a low-level binary search tree.
- **_gdsi_bstree_t _gdsi_bstree_insert**(**_gdsi_bstree_t** *T, const **gdsi_compare_func_t** COMP_F, const **gdsi_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 infix 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 postfix 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 Typedef Documentation

4.2.1.1 typedef `_gdsl_bintree_t _gdsl_bstree_t`

GDSL low-level binary search tree type.

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

Definition at line 52 of file `_gdsl_bstree.h`.

4.2.1.2 typedef `int(* _gdsl_bstree_map_func_t)(_gdsl_bstree_t TREE, void *USER_DATA)`

GDSL low-level binary search tree map function type.

Parameters

<i>TREE</i>	The low-level binary search tree to map.
<i>USER_DATA</i> <i>A</i>	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 61 of file `_gdsl_bstree.h`.

4.2.1.3 `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

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

Definition at line 71 of file `_gdsl_bstree.h`.

4.2.2 Function Documentation

4.2.2.1 `_gdsl_bstree_t _gdsl_bstree_alloc(const gdsl_element_t E)`

Create a new low-level binary search tree.

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

Note

Complexity: $O(1)$

Precondition

nothing.

Parameters

<i>E</i>	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. 30)

4.2.2.2 `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

<i>T</i>	The low-level binary search tree to destroy.
<i>FREE_F</i>	The function used to deallocate T's nodes contents.

See also

`_gdsl_bstree_alloc()` (p. 29)

4.2.2.3 `_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

<i>T</i>	The low-level binary search tree to copy.
<i>COPY_F</i>	The function used to copy T's nodes contents.

Returns

a copy of T in case of success.

NULL if `_gdsI_bstree_is_empty(T) == TRUE` or in case of insufficient memory.

See also

`_gdsI_bstree_alloc()` (p. 29)

`_gdsI_bstree_free()` (p. 30)

`_gdsI_bstree_is_empty()` (p. 31)

4.2.2.4 `bool _gdsI_bstree_is_empty(const _gdsI_bstree_t T)`

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

Note

Complexity: $O(1)$

Precondition

nothing.

Parameters

<i>T</i>	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

`_gdsI_bstree_is_leaf()` (p. 32)

`_gdsI_bstree_is_root()` (p. 33)

4.2.2.5 `bool _gdsi_bstree_is_leaf(const _gdsi_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 `_gdsi_bstree_t`.

Parameters

<code>T</code>	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

`_gdsi_bstree_is_empty()` (p. 31)
`_gdsi_bstree_is_root()` (p. 33)

4.2.2.6 `gdsi_element_t _gdsi_bstree_get_content(const _gdsi_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 `_gdsi_bstree_t`.

Parameters

<code>T</code>	The low-level binary search tree to use.
----------------	--

Returns

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

4.2.2.7 `bool _gdsi_bstree_is_root(const _gdsi_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 `_gdsi_bstree_t`.

Parameters

<code>T</code>	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

`_gdsi_bstree_is_empty()` (p. 31)
`_gdsi_bstree_is_leaf()` (p. 32)

4.2.2.8 `_gdsi_bstree_t _gdsi_bstree_get_parent(const _gdsi_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 `_gdsi_bstree_t`.

Parameters

<code>T</code>	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

`_gdsi_bstree_is_root()` (p. 33)

4.2.2.9 `_gdsi_bstree_t _gdsi_bstree_get_left(const _gdsi_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 `_gdsi_bstree_t`.

Parameters

<code>T</code>	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

`_gdsi_bstree_get_right()` (p. 34)

4.2.2.10 `_gdsi_bstree_t _gdsi_bstree_get_right(const _gdsi_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 `_gdsi_bstree_t`.

Parameters

<code>T</code>	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

`_gdsi_bstree_get_left()` (p. 34)

4.2.2.11 `ulong _gdsi_bstree_get_size(const _gdsi_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

`_gdsi_bstree_get_height()` (p. 35)

4.2.2.12 `ulong _gdsi_bstree_get_height(const _gdsi_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

<i>T</i>	The low-level binary search tree to compute the height from.
----------	--

Returns

the height of *T*.

See also

[`_gdsl_bstree_get_size\(\)`](#) (p. 35)

4.2.2.13 `_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 $\log_2(|T|) \leq h(T) \leq |T|-1$

Precondition

COMP_F != NULL & *RESULT* != NULL.

Parameters

<i>T</i>	The reference of the low-level binary search tree to use.
<i>COMP_F</i>	The comparison function to use to compare <i>T</i> 's elements with <i>VALUE</i> to find <i>E</i> .
<i>VALUE</i>	The value used to search for the element <i>E</i> .
<i>RESULT</i>	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. 37)

[`_gdsl_bstree_remove\(\)`](#) (p. 37)

4.2.2.14 `gdsi_element_t gdsi_bstree_remove(_gdsi_bstree_t * T, const gdsi_compare_func_t COMP_F, const gdsi_element_t VALUE)`

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

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

Note

Complexity: $O(h(T))$, where $\log_2(|T|) \leq h(T) \leq |T|-1$

The resulting T is modified by examining the left sub-tree from the founded e.

Precondition

COMP_F != NULL.

Parameters

<i>T</i>	The reference of the low-level binary search tree to modify.
<i>COMP_F</i>	The comparison function to use to compare T's elements with VALUE to find the element e to remove.
<i>VALUE</i>	The value that must be used by COMP_F to find the element e to remove.

Returns

the first founded element equal to VALUE in T.

NULL if no element equal to VALUE is found or if T is empty.

See also

`_gdsi_bstree_insert()` (p. 36)

`_gdsi_bstree_search()` (p. 37)

4.2.2.15 `_gdsi_bstree_t _gdsi_bstree_search(const _gdsi_bstree_t T, const gdsi_compare_func_t COMP_F, const gdsi_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 $\log_2(|T|) \leq h(T) \leq |T|-1$

Precondition

COMP_F != NULL.

Parameters

<i>T</i>	The low-level binary search tree to use.
<i>COMP_F</i>	The comparison function to use to compare T's elements with VALUE to find the element E.
<i>VALUE</i>	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

[_gdsi_bstree_insert\(\)](#) (p. 36)
[_gdsi_bstree_remove\(\)](#) (p. 37)

4.2.2.16 `_gdsi_bstree_t _gdsi_bstree_search_next(const _gdsi_bstree_t T, const gdsi_compare_func_t COMP_F, const gdsi_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 $\log_2(|T|) \leq h(T) \leq |T|-1$

Precondition

COMP_F != NULL.

Parameters

<i>T</i>	The low-level binary search tree to use.
<i>COMP_F</i>	The comparison function to use to compare T's elements with VALUE to find the element E.
<i>VALUE</i>	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.

4.2.2.17 `_gdsI_bstree_t _gdsI_bstree_map_prefix(const _gdsI_bstree_t T, const _gdsI_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 - GD_SL_MAP_STOP, then `_gdsI_bstree_map_prefix()` (p. 39) stops and returns its last examined node.

Note

Complexity: $O(|T|)$

Precondition

MAP_F != NULL.

Parameters

<i>T</i>	The low-level binary search tree to map.
<i>MAP_F</i>	The map function.
<i>USER_DATA</i>	User's datas passed to MAP_F.

Returns

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

See also

`_gdsI_bstree_map_infix()` (p. 39)
`_gdsI_bstree_map_postfix()` (p. 40)

4.2.2.18 `_gdsI_bstree_t _gdsI_bstree_map_infix(const _gdsI_bstree_t T, const _gdsI_bstree_map_func_t MAP_F, void * USER_DATA)`

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

Parse all nodes of the low-level binary search tree T in infix order. The MAP_F function is called on each node with the USER_DATA argument. If MAP_F returns - GD_SL_MAP_STOP, then `_gdsI_bstree_map_infix()` (p. 39) stops and returns its last examined node.

Note

Complexity: $O(|T|)$

Precondition

$MAP_F \neq NULL$.

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

[`_gdsi_bstree_map_prefix\(\)`](#) (p. 39)
[`_gdsi_bstree_map_postfix\(\)`](#) (p. 40)

4.2.2.19 `_gdsi_bstree_t _gdsi_bstree_map_postfix(const _gdsi_bstree_t T, const _gdsi_bstree_map_func_t MAP_F, void * USER_DATA)`

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

Parse all nodes of the low-level binary search tree T in postfix order. The MAP_F function is called on each node with the $USER_DATA$ argument. If MAP_F returns $GDSL_MAP_STOP$, then [`_gdsi_bstree_map_postfix\(\)`](#) (p. 40) stops and returns its last examined node.

Note

Complexity: $O(|T|)$

Precondition

$MAP_F \neq NULL$.

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 GDSDL_MAP_STOP.
 NULL when the parsing is done.

See also

`_gdsl_bstree_map_prefix()` (p. 39)
`_gdsl_bstree_map_infix()` (p. 39)

4.2.2.20 `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

<i>T</i>	The low-level binary search tree to write.
<i>WRITE_F</i>	The write function.
<i>OUTPUT_FILE</i>	The file where to write T's nodes.
<i>USER_DATA</i>	User's datas passed to WRITE_F.

See also

`_gdsl_bstree_write_xml()` (p. 41)
`_gdsl_bstree_dump()` (p. 42)

4.2.2.21 `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 WRITE_F.

Note

Complexity: $O(|T|)$

Precondition

OUTPUT_FILE != NULL.

Parameters

<i>T</i>	The low-level binary search tree to write.
<i>WRITE_F</i>	The write function.
<i>OUTPUT_FILE</i>	The file where to write T's nodes.
<i>USER_DATA</i>	User's datas passed to WRITE_F.

See also

`_gdsi_bstree_write()` (p. 41)
`_gdsi_bstree_dump()` (p. 42)

4.2.2.22 void `_gdsi_bstree_dump` (const `_gdsi_bstree_t` *T*, const `_gdsi_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

<i>T</i>	The low-level binary search tree to dump.
<i>WRITE_F</i>	The write function.
<i>OUTPUT_FILE</i>	The file where to write T's nodes.
<i>USER_DATA</i>	User's datas passed to WRITE_F.
<i>A</i>	Generated on Tue Aug 21 2012 16:00:00 for gdsi by Doxygen

See also

`_gdsi_bstree_write()` (p. 41)

`_gdsi_bstree_write_xml()` (p. 41)

4.3 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.

4.3.1 Typedef Documentation

4.3.1.1 typedef `_gdsl_node_t _gdsl_list_t`

GDSDL 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 54 of file `_gdsl_list.h`.

4.3.2 Function Documentation

4.3.2.1 `_gdsl_list_t _gdsl_list_alloc (const gdsl_element_t E)`

Create a new low-level list.

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

Note

Complexity: $O(1)$

Precondition

nothing.

Parameters

<code>E</code>	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. 45)

4.3.2.2 `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

<i>L</i>	The low-level list to destroy.
<i>FREE_F</i>	The function used to deallocated L's nodes contents.

See also

`_gdsl_list_alloc()` (p. 45)

4.3.2.3 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

<i>L</i>	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.

4.3.2.4 ulong _gdsl_list_get_size(const _gdsl_list_t L)

Get the size of a low-level list.

Note

Complexity: $O(|L|)$

Precondition

nothing.

Parameters

<i>L</i>	The low-level list to use.
----------	----------------------------

Returns

the number of elements of *L* (noted $|L|$).

4.3.2.5 void _gdsI_list_link(_gdsI_list_t *L1*, _gdsI_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 _gdsI_list_t.

Parameters

<i>L1</i>	The low-level list to link before <i>L2</i> .
<i>L2</i>	The low-level list to link after <i>L1</i> .

4.3.2.6 void _gdsI_list_insert_after(_gdsI_list_t *L*, _gdsI_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 _gdsI_list_t.

Parameters

<i>L</i>	The low-level list to link after <i>PREV</i> .
<i>PREV</i>	The low-level list that will be linked before <i>L</i> .

See also

`_gdsi_list_insert_before()` (p. 48)

`_gdsi_list_remove()` (p. 48)

4.3.2.7 void `_gdsi_list_insert_before(_gdsi_list_t L, _gdsi_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 `_gdsi_list_t`.

Parameters

<i>L</i>	The low-level list to link before SUCC.
<i>SUCC</i>	The low-level list that will be linked after L.

See also

`_gdsi_list_insert_after()` (p. 47)

`_gdsi_list_remove()` (p. 48)

4.3.2.8 void `_gdsi_list_remove(_gdsi_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 `_gdsi_node_t`.

Parameters

<i>NODE</i>	The low-level node to unlink from the low-level list in which it's linked.
-------------	--

See also

`_gdsl_list_insert_after()` (p. 47)
`_gdsl_list_insert_before()` (p. 48)

4.3.2.9 `_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

<i>L</i>	The low-level list to use
<i>COMP_F</i>	The comparison function to use to compare <i>L</i> 's elements with <i>VALUE</i> to find the element <i>e</i>
<i>VALUE</i>	The value that must be used by <i>COMP_F</i> to find the element <i>e</i>

Returns

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

4.3.2.10 `_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 *GDSDL_MAP_STOP*, then `_gdsl_list_map_forward()` (p. 49) stops and returns its last examined node.

Note

Complexity: $O(|L|)$

Precondition

MAP_F != NULL.

Parameters

<i>L</i>	Th low-level list to map.
<i>MAP_F</i>	The map function.
<i>USER_DATA</i>	User's datas.

Returns

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

See also

`_gdsl_list_map_backward()` (p. 50)

4.3.2.11 **`_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 GDSDL_MAP_STOP, then **`_gdsl_list_map_backward()`** (p. 50) stops and returns its last examined node.

Note

Complexity: $O(2 |L|)$

Precondition

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

Parameters

<i>L</i>	Th low-level list to map.
<i>MAP_F</i>	The map function.
<i>USER_DATA</i>	User's datas.

Returns

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

See also

[`_gdsi_list_map_forward\(\)`](#) (p. 49)

4.3.2.12 `void _gdsi_list_write(const _gdsi_list_t L, const _gdsi_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

<i>L</i>	The low-level list to write.
<i>WRITE_F</i>	The write function.
<i>OUTPUT_FILE</i>	The file where to write L's nodes.
<i>USER_DATA</i>	User's datas passed to WRITE_F.

See also

[`_gdsi_list_write_xml\(\)`](#) (p. 51)

[`_gdsi_list_dump\(\)`](#) (p. 52)

4.3.2.13 `void _gdsi_list_write_xml(const _gdsi_list_t L, const _gdsi_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

<i>L</i>	The low-level list to write.
<i>WRITE_F</i>	The write function.
<i>OUTPUT_FILE</i>	The file where to write L's nodes.
<i>USER_DATA</i>	User's datas passed to <i>WRITE_F</i> .

See also

`_gdsi_list_write()` (p. 51)
`_gdsi_list_dump()` (p. 52)

4.3.2.14 `void _gdsi_list_dump(const _gdsi_list_t L, const _gdsi_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

<i>L</i>	The low-level list to dump.
<i>WRITE_F</i>	The write function.
<i>OUTPUT_FILE</i>	The file where to write L's nodes.
<i>USER_DATA</i>	User's datas passed to <i>WRITE_F</i> .

See also

`_gdsi_list_write()` (p. 51)
`_gdsi_list_write_xml()` (p. 51)

4.4 Low-level doubly-linked node manipulation module

Typedefs

- typedef struct _gdsI_node * **_gdsI_node_t**
GDSL low-level doubly linked node type.
- typedef int(* **_gdsI_node_map_func_t**)(const **_gdsI_node_t** NODE, void *USER_DATA)
GDSL low-level doubly-linked node map function type.
- typedef void(* **_gdsI_node_write_func_t**)(const **_gdsI_node_t** NODE, FILE *OUTPUT_FILE, void *USER_DATA)
GDSL low-level doubly-linked node write function type.

Functions

- **_gdsI_node_t _gdsI_node_alloc** (void)
Create a new low-level node.
- **gdsI_element_t _gdsI_node_free** (_gdsI_node_t NODE)
Destroy a low-level node.
- **_gdsI_node_t _gdsI_node_get_succ** (const **_gdsI_node_t** NODE)
Get the successor of a low-level node.
- **_gdsI_node_t _gdsI_node_get_pred** (const **_gdsI_node_t** NODE)
Get the predecessor of a low-level node.
- **gdsI_element_t _gdsI_node_get_content** (const **_gdsI_node_t** NODE)
Get the content of a low-level node.
- void **_gdsI_node_set_succ** (_gdsI_node_t NODE, const **_gdsI_node_t** SUC-C)
Set the successor of a low-level node.
- void **_gdsI_node_set_pred** (_gdsI_node_t NODE, const **_gdsI_node_t** PRE-D)
Set the predecessor of a low-level node.
- void **_gdsI_node_set_content** (_gdsI_node_t NODE, const **gdsI_element_t** -CONTENT)
Set the content of a low-level node.
- void **_gdsI_node_link** (_gdsI_node_t NODE1, **_gdsI_node_t** NODE2)
Link two low-level nodes together.
- void **_gdsI_node_unlink** (_gdsI_node_t NODE1, **_gdsI_node_t** NODE2)
Unlink two low-level nodes.
- void **_gdsI_node_write** (const **_gdsI_node_t** NODE, const **_gdsI_node_write_func_t** WRITE_F, FILE *OUTPUT_FILE, void *USER_DATA)
Write a low-level node to a file.
- void **_gdsI_node_write_xml** (const **_gdsI_node_t** NODE, const **_gdsI_node_write_func_t** WRITE_F, FILE *OUTPUT_FILE, void *USER_DATA)
Write a low-level node to a file into XML.

- `void _gdsI_node_dump (const _gdsI_node_t NODE, const _gdsI_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 Typedef Documentation

4.4.1.1 `typedef struct _gdsI_node* _gdsI_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 53 of file `_gdsI_node.h`.

4.4.1.2 `typedef int(* _gdsI_node_map_func_t)(const _gdsI_node_t NODE, void *USER_DATA)`

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

Parameters

<code>NODE</code>	The low-level node to map.
<code>USER_DATA</code>	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 62 of file `_gdsI_node.h`.

4.4.1.3 `typedef void(* _gdsI_node_write_func_t)(const _gdsI_node_t NODE, FILE *OUTPUT_FILE, void *USER_DATA)`

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

Parameters

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

Definition at line 72 of file `_gdsI_node.h`.

4.4.2 Function Documentation

4.4.2.1 `_gdsl_node_t _gdsl_node_alloc(void)`

Create a new low-level node.

Allocate a new low-level node data structure.

Note

Complexity: $O(1)$

Precondition

nothing.

Returns

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

See also

`_gdsl_node_free()` (p. 55)

4.4.2.2 `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. 55)

4.4.2.3 `_gdsi_node_t _gdsi_node_get_succ(const _gdsi_node_t NODE)`

Get the successor of a low-level node.

Note

Complexity: $O(1)$

Precondition

`NODE != NULL`

Parameters

<code>NODE</code>	The low-level node which we want to get the successor from.
-------------------	---

Returns

the successor of the low-level node `NODE` if `NODE` has a successor.
NULL if the low-level node `NODE` has no successor.

See also

`_gdsi_node_get_pred()` (p. 56)

`_gdsi_node_set_succ()` (p. 57)

`_gdsi_node_set_pred()` (p. 58)

4.4.2.4 `_gdsi_node_t _gdsi_node_get_pred(const _gdsi_node_t NODE)`

Get the predecessor of a low-level node.

Note

Complexity: $O(1)$

Precondition

`NODE != NULL`

Parameters

<code>NODE</code>	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. 56)

[`_gdsl_node_set_succ\(\)`](#) (p. 57)

[`_gdsl_node_set_pred\(\)`](#) (p. 58)

4.4.2.5 `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

<code>NODE</code>	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. 58)

4.4.2.6 `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

<i>NODE</i>	The low-level node which want to change the successor from.
<i>SUCC</i>	The new successor of <i>NODE</i> .

See also

`_gdsI_node_get_succ()` (p. 56)

4.4.2.7 `void _gdsI_node_set_pred(_gdsI_node_t NODE, const _gdsI_node_t PRED)`

Set the predecessor of a low-level node.

Modify the predecessor of the low-level node *NODE* to *PRED*.

Note

Complexity: $O(1)$

Precondition

NODE != NULL

Parameters

<i>NODE</i>	The low-level node which want to change the predecessor from.
<i>PRED</i>	The new predecessor of <i>NODE</i> .

See also

`_gdsI_node_get_pred()` (p. 56)

4.4.2.8 `void _gdsI_node_set_content(_gdsI_node_t NODE, const gdsI_element_t CONTENT)`

Set the content of a low-level node.

Modify the content of the low-level node *NODE* to *CONTENT*.

Note

Complexity: $O(1)$

Precondition

NODE != NULL

Parameters

<i>NODE</i>	The low-level node which want to change the content from.
<i>CONTENT</i>	The new content of <i>NODE</i> .

See also

[`_gdsI_node_get_content\(\)`](#) (p. 57)

4.4.2.9 void `_gdsI_node_link(_gdsI_node_t NODE1, _gdsI_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

<i>NODE1</i>	The first low-level node to link to <i>NODE2</i> .
<i>NODE2</i>	The second low-level node to link from <i>NODE1</i> .

See also

[`_gdsI_node_unlink\(\)`](#) (p. 59)

4.4.2.10 void `_gdsI_node_unlink(_gdsI_node_t NODE1, _gdsI_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

<i>NODE1</i>	The first low-level node to unlink from <i>NODE2</i> .
<i>NODE2</i>	The second low-level node to unlink from <i>NODE1</i> .

See also

[`_gdsi_node_link\(\)`](#) (p. 59)

4.4.2.11 `void _gdsi_node_write (const _gdsi_node_t NODE, const
_gdsi_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

<i>NODE</i>	The low-level node to write.
<i>WRITE_F</i>	The write function.
<i>OUTPUT_FILE</i>	The file where to write <i>NODE</i> .
<i>USER_DATA</i>	User's datas passed to <i>WRITE_F</i> .

See also

[`_gdsi_node_write_xml\(\)`](#) (p. 60)

[`_gdsi_node_dump\(\)`](#) (p. 61)

4.4.2.12 `void _gdsi_node_write_xml (const _gdsi_node_t NODE, const
_gdsi_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

<i>NODE</i>	The low-level node to write.
<i>WRITE_F</i>	The write function.
<i>OUTPUT_FILE</i>	The file where to write <i>NODE</i> .
<i>USER_DATA</i>	User's datas passed to <i>WRITE_F</i> .

See also

`_gdsI_node_write()` (p. 60)

`_gdsI_node_dump()` (p. 61)

4.4.2.13 `void _gdsI_node_dump (const _gdsI_node_t NODE, const
_gdsI_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

<i>NODE</i>	The low-level node to dump.
<i>WRITE_F</i>	The write function.
<i>OUTPUT_FILE</i>	The file where to write <i>NODE</i> .
<i>USER_DATA</i>	User's datas passed to <i>WRITE_F</i> .

See also

[_gdsi_node_write\(\)](#) (p. 60)

[_gdsi_node_write_xml\(\)](#) (p. 60)

4.5 Main module

Functions

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

4.5.1 Function Documentation

4.5.1.1 `const char* gdsI_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

Typedefs

- typedef struct gdsi_2darray * **gdsi_2darray_t**
GDSL 2D-array type.

Functions

- **gdsi_2darray_t gdsi_2darray_alloc** (const char *NAME, const **ulong** R, const **ulong** C, const **gdsi_alloc_func_t** ALLOC_F, const **gdsi_free_func_t** FREE_F)
Create a new 2D-array.
- void **gdsi_2darray_free** (**gdsi_2darray_t** A)
Destroy a 2D-array.
- const char * **gdsi_2darray_get_name** (const **gdsi_2darray_t** A)
Get the name of a 2D-array.
- **ulong gdsi_2darray_get_rows_number** (const **gdsi_2darray_t** A)
Get the number of rows of a 2D-array.
- **ulong gdsi_2darray_get_columns_number** (const **gdsi_2darray_t** A)
Get the number of columns of a 2D-array.
- **ulong gdsi_2darray_get_size** (const **gdsi_2darray_t** A)
Get the size of a 2D-array.
- **gdsi_element_t gdsi_2darray_get_content** (const **gdsi_2darray_t** A, const **ulong** R, const **ulong** C)
Get an element from a 2D-array.
- **gdsi_2darray_t gdsi_2darray_set_name** (**gdsi_2darray_t** A, const char *NEW_NAME)
Set the name of a 2D-array.
- **gdsi_element_t gdsi_2darray_set_content** (**gdsi_2darray_t** A, const **ulong** R, const **ulong** C, void *VALUE)
Modify an element in a 2D-array.
- void **gdsi_2darray_write** (const **gdsi_2darray_t** A, const **gdsi_write_func_t** WRITE_F, FILE *OUTPUT_FILE, void *USER_DATA)
Write the content of a 2D-array to a file.
- void **gdsi_2darray_write_xml** (const **gdsi_2darray_t** A, const **gdsi_write_func_t** WRITE_F, FILE *OUTPUT_FILE, void *USER_DATA)
Write the content of a 2D array to a file into XML.
- void **gdsi_2darray_dump** (const **gdsi_2darray_t** A, const **gdsi_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 Typedef Documentation

4.6.1.1 typedef struct gdsl_2darray* gdsl_2darray_t

GDSDL 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 53 of file gdsl_2darray.h.

4.6.2 Function Documentation

4.6.2.1 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

<i>NAME</i>	The name of the new 2D-array to create
<i>R</i>	The number of rows of the new 2D-array to create
<i>C</i>	The number of columns of the new 2D-array to create
<i>ALLOC_F</i>	Function to alloc element when inserting it in a 2D-array
<i>FREE_F</i>	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

gdsI_2darray_free() (p. 66)

gdsI_alloc_func_t (p. 223)

gdsI_free_func_t (p. 223)

4.6.2.2 void gdsI_2darray_free(gdsI_2darray_t A)

Destroy a 2D-array.

Flush and destroy the 2D-array A. The FREE_F function passed to **gdsI_2darray_alloc()** (p. 65) 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: $O(R \times C)$, where R is A's rows count, and C is A's columns count

Precondition

A must be a valid gdsI_2darray_t

Parameters

A	The 2D-array to destroy
---	-------------------------

See also

gdsI_2darray_alloc() (p. 65)

4.6.2.3 const char* gdsI_2darray_get_name(const gdsI_2darray_t A)

Get the name of a 2D-array.

Note

Complexity: $O(1)$

Precondition

A must be a valid gdsI_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

gdsI_2darray_set_name() (p. 69)

4.6.2.4 `ulong gdsI_2darray_get_rows_number(const gdsI_2darray_t A)`

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

Note

Complexity: $O(1)$

Precondition

A must be a valid `gdsI_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

gdsI_2darray_get_columns_number() (p. 67)

gdsI_2darray_get_size() (p. 68)

4.6.2.5 `ulong gdsI_2darray_get_columns_number(const gdsI_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. 67)

`gdsl_2darray_get_size()` (p. 68)

4.6.2.6 `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. 67)

`gdsl_2darray_get_columns_number()` (p. 67)

4.6.2.7 `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 `gdsi_2darray_t` & $R \leq \text{gdsi_2darray_get_rows_number}(A)$ & $C \leq \text{gdsi_2darray_get_columns_number}(A)$

Parameters

<i>A</i>	The 2D-array from which getting the element
<i>R</i>	The row index of the element to get
<i>C</i>	The column index of the element to get

Returns

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

See also

`gdsi_2darray_set_content()` (p. 70)

4.6.2.8 `gdsi_2darray_t gdsi_2darray_set_name(gdsi_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 `gdsi_2darray_t`

Parameters

<i>A</i>	The 2D-array to change the name
<i>NEW_NAME</i>	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. 66)

4.6.2.9 **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 & $R \leq \text{gdsl_2darray_get_rows_number}(A)$ & $C \leq \text{gdsl_2darray_get_columns_number}(A)$

Parameters

<i>A</i>	The 2D-array to modify on element from
<i>R</i>	The row number of the element to modify
<i>C</i>	The column number of the element to modify
<i>VALUE</i>	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. 68)

4.6.2.10 **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: $O(R \times C)$, where *R* is *A*'s rows count, and *C* is *A*'s columns count

Precondition

WRITE_F != NULL & *OUTPUT_FILE* != NULL

Parameters

<i>A</i>	The 2D-array to write
<i>WRITE_F</i>	The write function
<i>OUTPUT_FILE</i>	The file where to write <i>A</i> 's elements
<i>USER_DATA</i>	User's datas passed to <i>WRITE_F</i>

See also

gdsi_2darray_write_xml() (p. 71)

gdsi_2darray_dump() (p. 72)

4.6.2.11 void **gdsi_2darray_write_xml**(const **gdsi_2darray_t** *A*, const
gdsi_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: $O(R \times C)$, where *R* is *A*'s rows count, and *C* is *A*'s columns count

Precondition

A must be a valid **gdsi_2darray_t** & *OUTPUT_FILE* != NULL

Parameters

<i>A</i>	The 2D-array to write
<i>WRITE_F</i>	The write function
<i>OUTPUT_FILE</i>	The file where to write <i>A</i> 's elements
<i>USER_DATA</i>	User's datas passed to <i>WRITE_F</i>

See also

gdsI_2darray_write() (p. 70)
gdsI_2darray_dump() (p. 72)

4.6.2.12 **void gdsI_2darray_dump(const gdsI_2darray_t A, const gdsI_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: $O(R \times C)$, where R is A's rows count, and C is A's columns count

Precondition

A must be a valid gdsI_2darray_t & OUTPUT_FILE != NULL

Parameters

<i>A</i>	The 2D-array to dump
<i>WRITE_F</i>	The write function
<i>OUTPUT_FILE</i>	The file where to write A's elements
<i>USER_DATA</i>	User's datas passed to WRITE_F

See also

gdsI_2darray_write() (p. 70)
gdsI_2darray_write_xml() (p. 71)

4.7 Binary search tree manipulation module

Typedefs

- typedef struct gds_l_bstree * **gds_l_bstree_t**
GDSL binary search tree type.

Functions

- **gds_l_bstree_t gds_l_bstree_alloc** (const char *NAME, **gds_l_alloc_func_t** ALL-OC_F, **gds_l_free_func_t** FREE_F, **gds_l_compare_func_t** COMP_F)
Create a new binary search tree.
- void **gds_l_bstree_free** (**gds_l_bstree_t** T)
Destroy a binary search tree.
- void **gds_l_bstree_flush** (**gds_l_bstree_t** T)
Flush a binary search tree.
- const char * **gds_l_bstree_get_name** (const **gds_l_bstree_t** T)
Get the name of a binary search tree.
- bool **gds_l_bstree_is_empty** (const **gds_l_bstree_t** T)
Check if a binary search tree is empty.
- **gds_l_element_t gds_l_bstree_get_root** (const **gds_l_bstree_t** T)
Get the root of a binary search tree.
- **ulong gds_l_bstree_get_size** (const **gds_l_bstree_t** T)
Get the size of a binary search tree.
- **ulong gds_l_bstree_get_height** (const **gds_l_bstree_t** T)
Get the height of a binary search tree.
- **gds_l_bstree_t gds_l_bstree_set_name** (**gds_l_bstree_t** T, const char *NEW_NAME)
Set the name of a binary search tree.
- **gds_l_element_t gds_l_bstree_insert** (**gds_l_bstree_t** T, void *VALUE, int *RESULT)
Insert an element into a binary search tree if it's not found or return it.
- **gds_l_element_t gds_l_bstree_remove** (**gds_l_bstree_t** T, void *VALUE)
Remove an element from a binary search tree.
- **gds_l_bstree_t gds_l_bstree_delete** (**gds_l_bstree_t** T, void *VALUE)
Delete an element from a binary search tree.
- **gds_l_element_t gds_l_bstree_search** (const **gds_l_bstree_t** T, **gds_l_compare_func_t** COMP_F, void *VALUE)
Search for a particular element into a binary search tree.
- **gds_l_element_t gds_l_bstree_map_prefix** (const **gds_l_bstree_t** T, **gds_l_map_func_t** MAP_F, void *USER_DATA)
Parse a binary search tree in prefixed order.
- **gds_l_element_t gds_l_bstree_map_infix** (const **gds_l_bstree_t** T, **gds_l_map_func_t** MAP_F, void *USER_DATA)

Parse a binary search tree in infix 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 postfix order.

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

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

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

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

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

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

4.7.1 Typedef Documentation

4.7.1.1 typedef struct **gdsl_bstree*** **gdsl_bstree_t**

GDSL binary search tree type.

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

Definition at line 53 of file **gdsl_bstree.h**.

4.7.2 Function Documentation

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

Create a new binary search tree.

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

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

Note

Complexity: $O(1)$

Precondition

nothing

Parameters

<i>NAME</i>	The name of the new binary tree to create
<i>ALLOC_F</i>	Function to alloc element when inserting it in a binary tree
<i>FREE_F</i>	Function to free element when removing it from a binary tree
<i>COMP_F</i>	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

gdsI_bstree_free() (p. 75)
gdsI_bstree_flush() (p. 76)
gdsI_alloc_func_t (p. 223)
gdsI_free_func_t (p. 223)
gdsI_compare_func_t (p. 224)

4.7.2.2 void gdsI_bstree_free (gdsI_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 **gdsI_bstree_alloc()** (p. 74). The name of T is deallocated and T is deallocated itself too.

Note

Complexity: $O(|T|)$

Precondition

T must be a valid gdsI_bstree_t

Parameters

<i>T</i>	The binary search tree to deallocate
----------	--------------------------------------

See also

gdsI_bstree_alloc() (p. 74)
gdsI_bstree_flush() (p. 76)

4.7.2.3 void `gdsI_bstree_flush`(`gdsI_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 **`gdsI_rbtrees_alloc()`** (p. 194). 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 `gdsI_bstree_t`

Parameters

<code>T</code>	The binary search tree to flush
----------------	---------------------------------

See also

`gdsI_bstree_alloc()` (p. 74)

`gdsI_bstree_free()` (p. 75)

4.7.2.4 const char* `gdsI_bstree_get_name`(const `gdsI_bstree_t T`)

Get the name of a binary search tree.

Note

Complexity: $O(1)$

Precondition

`T` must be a valid `gdsI_bstree_t`

Postcondition

The returned string MUST NOT be freed.

Parameters

<code>T</code>	The binary search tree to get the name from
----------------	---

Returns

the name of the binary search tree T.

See also

gdsI_bstree_set_name (p. 79) ()

4.7.2.5 bool gdsI_bstree_is_empty(const gdsI_bstree_t T)

Check if a binary search tree is empty.

Note

Complexity: $O(1)$

Precondition

T must be a valid gdsI_bstree_t

Parameters

<i>T</i>	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.

4.7.2.6 gdsI_element_t gdsI_bstree_get_root(const gdsI_bstree_t T)

Get the root of a binary search tree.

Note

Complexity: $O(1)$

Precondition

T must be a valid gdsI_bstree_t

Parameters

<i>T</i>	The binary search tree to get the root element from
----------	---

Returns

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

4.7.2.7 `ulong gdsI_bstree_get_size(const gdsI_bstree_t T)`

Get the size of a binary search tree.

Note

Complexity: $O(1)$

Precondition

T must be a valid `gdsI_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

`gdsI_bstree_get_height()` (p. 78)

4.7.2.8 `ulong gdsI_bstree_get_height(const gdsI_bstree_t T)`

Get the height of a binary search tree.

Note

Complexity: $O(|T|)$

Precondition

T must be a valid `gdsI_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

gdsI_bstree_get_size() (p. 78)

4.7.2.9 gdsI_bstree_t gdsI_bstree_set_name(gdsI_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 `gdsI_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

gdsI_bstree_get_name() (p. 76)

4.7.2.10 gdsI_element_t gdsI_bstree_insert(gdsI_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 `gdsI_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 `gdsI_bstree_alloc` and is inserted and then returned.

Note

Complexity: $O(h(T))$, where $\log_2(|T|) \leq h(T) \leq |T|-1$

Precondition

T must be a valid `gdsi_bstree_t` & `RESULT != NULL`

Parameters

<i>T</i>	The binary search tree to modify
<i>VALUE</i>	The value used to make the new element to insert into T
<i>RESULT</i>	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

`gdsi_bstree_remove()` (p. 80)

`gdsi_bstree_delete()` (p. 81)

4.7.2.11 `gdsi_element_t gdsi_bstree_remove(gdsi_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 **`gdsi_bstree_alloc()`** (p. 74). If E is found, it is removed from T and then returned.

Note

Complexity: $O(h(T))$, where $\log_2(|T|) \leq h(T) \leq |T|-1$
 The resulting T is modified by examining the left sub-tree from the founded E.

Precondition

T must be a valid `gdsi_bstree_t`

Parameters

<i>T</i>	The binary search tree to modify
<i>VALUE</i>	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

gdsI_bstree_insert() (p. 79)
gdsI_bstree_delete() (p. 81)

4.7.2.12 gdsI_bstree_t gdsI_bstree_delete(gdsI_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 **gdsI_bstree_alloc()** (p. 74). If `E` is found, it is removed from `T` and `E` is deallocated using `T`'s `FREE_F` function passed to **gdsI_bstree_alloc()** (p. 74), then `T` is returned.

Note

Complexity: $O(h(T))$, where $\log_2(|T|) \leq h(T) \leq |T|-1$
 the resulting `T` is modified by examining the left sub-tree from the founded `E`.

Precondition

`T` must be a valid `gdsI_bstree_t`

Parameters

<code>T</code>	The binary search tree to remove an element from
<code>VALUE</code>	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

gdsI_bstree_insert() (p. 79)
gdsI_bstree_remove() (p. 80)

4.7.2.13 gdsI_element_t gdsI_bstree_search(const gdsI_bstree_t T, gdsI_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 search tree T, by using COMP_F function to find it. If COMP_F == NULL, then the COMP_F function passed to **gdsi_bstree_alloc()** (p. 74) is used.

Note

Complexity: $O(h(T))$, where $\log_2(|T|) \leq h(T) \leq |T|-1$

Precondition

T must be a valid gdsi_bstree_t

Parameters

<i>T</i>	The binary search tree to use.
<i>COMP_F</i>	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)
<i>VALUE</i>	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

gdsi_bstree_insert() (p. 79)
gdsi_bstree_remove() (p. 80)
gdsi_bstree_delete() (p. 81)

4.7.2.14 gdsi_element_t gdsi_bstree_map_prefix(const gdsi_bstree_t T, gdsi_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 **gdsi_bstree_map_prefix()** (p. 82) stops and returns its last examined element.

Note

Complexity: $O(|T|)$

Precondition

T must be a valid gdsi_bstree_t & MAP_F != NULL

Parameters

<i>T</i>	The binary search tree to map.
<i>MAP_F</i>	The map function.
<i>USER_DATA</i> - <i>A</i>	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. 83)
gdsl_bstree_map_postfix() (p. 84)

4.7.2.15 **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 infix order.

Parse all nodes of the binary search tree *T* in infix 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. 83) stops and returns its last examined element.

Note

Complexity: $O(|T|)$

Precondition

T must be a valid **gdsl_bstree_t** & MAP_F != NULL

Parameters

<i>T</i>	The binary search tree to map.
<i>MAP_F</i>	The map function.
<i>USER_DATA</i> - <i>A</i>	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. 82)
gdsl_bstree_map_postfix() (p. 84)

4.7.2.16 `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 postfix order.

Parse all nodes of the binary search tree T in postfix 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. 84) stops and returns its last examined element.

Note

Complexity: $O(|T|)$

Precondition

T must be a valid `gdsl_bstree_t` & MAP_F != NULL

Parameters

<i>T</i>	The binary search tree to map.
<i>MAP_F</i>	The map function.
<i>USER_DATA</i>	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. 82)
gdsl_bstree_map_infix() (p. 83)

4.7.2.17 `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 `gdsi_bstree_t` & `WRITE_F` != NULL & `OUTPUT_FILE` != NULL

Parameters

<i>T</i>	The binary search tree to write.
<i>WRITE_F</i>	The write function.
<i>OUTPUT_FILE</i>	The file where to write T's elements.
<i>USER_DATA</i>	User's datas passed to <code>WRITE_F</code> .

See also

`gdsi_bstree_write_xml()` (p. 85)

`gdsi_bstree_dump()` (p. 86)

4.7.2.18 `void gdsi_bstree_write_xml(const gdsi_bstree_t T, gdsi_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 `gdsi_bstree_t` & `OUTPUT_FILE` != NULL

Parameters

<i>T</i>	The binary search tree to write.
<i>WRITE_F</i>	The write function.
<i>OUTPUT_FILE</i>	The file where to write T's elements.
<i>USER_DATA</i>	User's datas passed to <code>WRITE_F</code> .

See also

gdsI_bstree_write() (p. 84)

gdsI_bstree_dump() (p. 86)

**4.7.2.19 void gdsI_bstree_dump (const gdsI_bstree_t T, gdsI_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. Additional USER_DATA argument could be passed to WRITE_F.

Note

Complexity: $O(|T|)$

Precondition

T must be a valid gdsI_bstree_t & OUTPUT_FILE != NULL

Parameters

<i>T</i>	The binary search tree to write.
<i>WRITE_F</i>	The write function.
<i>OUTPUT_FILE</i>	The file where to write T's elements.
<i>USER_DATA</i>	User's datas passed to WRITE_F.

See also

gdsI_bstree_write() (p. 84)

gdsI_bstree_write_xml() (p. 85)

4.8 Hashtable manipulation module

Typedefs

- typedef struct hash_table * **gdsi_hash_t**
GDSL hashtable type.
- typedef const char *(* **gdsi_key_func_t**)(void *VALUE)
GDSL hashtable key function type.
- typedef **ulong**(* **gdsi_hash_func_t**)(const char *KEY)
GDSL hashtable hash function type.

Functions

- **ulong gdsi_hash** (const char *KEY)
Computes a hash value from a NULL terminated character string.
- **gdsi_hash_t gdsi_hash_alloc** (const char *NAME, **gdsi_alloc_func_t** ALLOC_F, **gdsi_free_func_t** FREE_F, **gdsi_key_func_t** KEY_F, **gdsi_hash_func_t** HASH_F, **ushort** INITIAL_ENTRIES_NB)
Create a new hashtable.
- void **gdsi_hash_free** (**gdsi_hash_t** H)
Destroy a hashtable.
- void **gdsi_hash_flush** (**gdsi_hash_t** H)
Flush a hashtable.
- const char * **gdsi_hash_get_name** (const **gdsi_hash_t** H)
Get the name of a hashtable.
- **ushort gdsi_hash_get_entries_number** (const **gdsi_hash_t** H)
Get the number of entries of a hashtable.
- **ushort gdsi_hash_get_lists_max_size** (const **gdsi_hash_t** H)
Get the max number of elements allowed in each entry of a hashtable.
- **ushort gdsi_hash_get_longest_list_size** (const **gdsi_hash_t** H)
Get the number of elements of the longest list entry of a hashtable.
- **ulong gdsi_hash_get_size** (const **gdsi_hash_t** H)
Get the size of a hashtable.
- double **gdsi_hash_get_fill_factor** (const **gdsi_hash_t** H)
Get the fill factor of a hashtable.
- **gdsi_hash_t gdsi_hash_set_name** (**gdsi_hash_t** H, const char *NEW_NAME)
Set the name of a hashtable.
- **gdsi_element_t gdsi_hash_insert** (**gdsi_hash_t** H, void *VALUE)
Insert an element into a hashtable (PUSH).
- **gdsi_element_t gdsi_hash_remove** (**gdsi_hash_t** H, const char *KEY)
Remove an element from a hashtable (POP).
- **gdsi_hash_t gdsi_hash_delete** (**gdsi_hash_t** H, const char *KEY)

Delete an element from a hashtable.

- **gdsl_hash_t gsdl_hash_modify** (gsdl_hash_t H, ushort NEW_ENTRIES_NB, ushort NEW_LISTS_MAX_SIZE)

Increase the dimensions of a hashtable.

- **gsdl_element_t gsdl_hash_search** (const gsdl_hash_t H, const char *KEY)

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

- **gsdl_element_t gsdl_hash_map** (const gsdl_hash_t H, gsdl_map_func_t MAP_F, void *USER_DATA)

Parse a hashtable.

- void **gsdl_hash_write** (const gsdl_hash_t H, gsdl_write_func_t WRITE_F, FILE *OUTPUT_FILE, void *USER_DATA)

Write all the elements of a hashtable to a file.

- void **gsdl_hash_write_xml** (const gsdl_hash_t H, gsdl_write_func_t WRITE_F, FILE *OUTPUT_FILE, void *USER_DATA)

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

- void **gsdl_hash_dump** (const gsdl_hash_t H, gsdl_write_func_t WRITE_F, FILE *OUTPUT_FILE, void *USER_DATA)

Dump the internal structure of a hashtable to a file.

4.8.1 Typedef Documentation

4.8.1.1 typedef struct hash_table* gsdl_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 54 of file gsdl_hash.h.

4.8.1.2 typedef const char*(* gsdl_key_func_t)(void *VALUE)

GDSL hashtable key function type.

Postcondition

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

Parameters

<i>VALUE</i>	The value used to get the key from
--------------	------------------------------------

Returns

The key associated to the VALUE.

Definition at line 62 of file gsdl_hash.h.

4.8.1.3 typedef ulong(* gdsI_hash_func_t)(const char *KEY)

GDSL hashtable hash function type.

Parameters

<i>KEY</i>	the key used to compute the hash code.
------------	--

Returns

The hashed value computed from KEY.

Definition at line 70 of file gdsI_hash.h.

4.8.2 Function Documentation

4.8.2.1 ulong gdsI_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

<i>KEY</i>	The NULL terminated string to compute the key from
------------	--

Returns

the hash code computed from KEY.

4.8.2.2 gdsI_hash_t gdsI_hash_alloc (const char * NAME, gdsI_alloc_func_t ALLOC_F, gdsI_free_func_t FREE_F, gdsI_key_func_t KEY_F, gdsI_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 **gdsI_hash_modify()** (p.98) 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 **gdsI_hash()** (p. 89) above

Note

Complexity: $O(1)$

Precondition

nothing.

Parameters

<i>NAME</i>	The name of the new hashtable to create
<i>ALLOC_F</i>	Function to alloc element when inserting it in the hashtable
<i>FREE_F</i>	Function to free element when deleting it from the hashtable
<i>KEY_F</i>	Function to get the key from an element
<i>HASH_F</i>	Function used to compute the hash value.
<i>INITIAL_ENTRIES_NB</i>	Initial number of entries of the hashtable

Returns

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

See also

gdsI_hash_free() (p. 90)
gdsI_hash_flush() (p. 91)
gdsI_hash_insert() (p. 96)
gdsI_hash_modify() (p. 98)

4.8.2.3 void gdsI_hash_free(gdsI_hash_t H)

Destroy a hashtable.

Deallocate all the elements of the hashtable *H* by calling *H*'s `FREE_F` function passed to **gdsI_hash_alloc()** (p. 89). The name of *H* is deallocated and *H* is deallocated itself too.

Note

Complexity: $O(|H|)$

Precondition

H must be a valid `gdsI_hash_t`

Parameters

<i>H</i>	The hashtable to destroy
----------	--------------------------

See also

gdsI_hash_alloc() (p. 89)

gdsI_hash_flush() (p. 91)

4.8.2.4 void gdsI_hash_flush(gdsI_hash_t *H*)

Flush a hashtable.

Deallocate all the elements of the hashtable *H* by calling *H*'s `FREE_F` function passed to **gdsI_hash_alloc()** (p. 89). *H* is not deallocated itself and *H*'s name is not modified.

Note

Complexity: $O(|H|)$

Precondition

H must be a valid `gdsI_hash_t`

Parameters

<i>H</i>	The hashtable to flush
----------	------------------------

See also

gdsI_hash_alloc() (p. 89)

gdsI_hash_free() (p. 90)

4.8.2.5 `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

<i>H</i>	The hashtable to get the name from
----------	------------------------------------

Returns

the name of the hashtable H.

See also

`gdsl_hash_set_name()` (p. 95)

4.8.2.6 `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

<i>H</i>	The hashtable to use.
----------	-----------------------

Returns

the number of lists entries of the hashtable H.

See also

gdsI_hash_get_size() (p. 94)
gdsI_hash_fill_factor()

4.8.2.7 ushort gdsI_hash_get_lists_max_size(const gdsI_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 `gdsI_hash_t`

Parameters

<i>H</i>	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 **gdsI_hash_modify()** (p. 98) was used with a `NEW_LISTS_MAX_SIZE` greather than the actual one.

See also

gdsI_hash_fill_factor()
gdsI_hash_get_entries_number() (p. 92)
gdsI_hash_get_longest_list_size() (p. 93)
gdsI_hash_modify() (p. 98)

4.8.2.8 ushort gdsI_hash_get_longest_list_size(const gdsI_hash_t H)

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

Note

Complexity: $O(L)$, where $L = \text{gdsI_hash_get_entries_number}(H)$

Precondition

H must be a valid `gdsI_hash_t`

Parameters

<i>H</i>	The hashtable to use.
----------	-----------------------

Returns

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

See also

`gdsI_hash_get_size()` (p. 94)
`gdsI_hash_fill_factor()`
`gdsI_hash_get_entries_number()` (p. 92)
`gdsI_hash_get_lists_max_size()` (p. 93)

4.8.2.9 `ulong gdsI_hash_get_size(const gdsI_hash_t H)`

Get the size of a hashtable.

Note

Complexity: $O(L)$, where $L = \text{gdsI_hash_get_entries_number}(H)$

Precondition

H must be a valid `gdsI_hash_t`

Parameters

<i>H</i>	The hashtable to get the size from
----------	------------------------------------

Returns

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

See also

`gdsI_hash_get_entries_number()` (p. 92)
`gdsI_hash_fill_factor()`
`gdsI_hash_get_longest_list_size()` (p. 93)

4.8.2.10 `double gdsI_hash_get_fill_factor(const gdsI_hash_t H)`

Get the fill factor of a hashtable.

Note

Complexity: $O(L)$, where $L = \text{gdsI_hash_get_entries_number}(H)$

Precondition

H must be a valid `gdsI_hash_t`

Parameters

<i>H</i>	The hashtable to use
----------	----------------------

Returns

The fill factor of H, computed as $|H| / L$

See also

`gdsI_hash_get_entries_number()` (p. 92)
`gdsI_hash_get_longest_list_size()` (p. 93)
`gdsI_hash_get_size()` (p. 94)

4.8.2.11 `gdsI_hash_t gdsI_hash_set_name(gdsI_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 `gdsI_hash_t`

Parameters

<i>H</i>	The hashtable to change the name
<i>NEW_NAME</i>	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. 92)

4.8.2.12 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 $HASH_F(K) \bmod gdsl_hash_get_entries_number(H)$. ALLOC_F, KEY_F and HASH_F are the function pointers passed to **gdsl_hash_alloc()** (p. 89).

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

<i>H</i>	The hashtable to modify
<i>VALUE</i>	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

gdsI_hash_alloc() (p. 89)
gdsI_hash_remove() (p. 97)
gdsI_hash_delete() (p. 97)
gdsI_hash_get_size() (p. 94)
gdsI_hash_get_entries_number() (p. 92)
gdsI_hash_modify() (p. 98)

4.8.2.13 **gdsI_element_t gdsI_hash_remove(gdsI_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 `gdsI_hash_t`

Parameters

<i>H</i>	The hashtable to modify
<i>KEY</i>	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

gdsI_hash_insert() (p. 96)
gdsI_hash_search() (p. 99)
gdsI_hash_delete() (p. 97)

4.8.2.14 **gdsI_hash_t gdsI_hash_delete(gdsI_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 **gdsI_hash_alloc()** (p. 89), 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. 96)
`gdsl_hash_search()` (p. 99)
`gdsl_hash_remove()` (p. 97)

4.8.2.15 `gdsl_hash_t gdsl_hash_modify (gdsl_hash_t H , ushort $NEW_ENTRIES_NB$, ushort $NEW_LISTS_MAX_SIZE$)`

Increase the dimensions of a hashtable.

The hashtable H is re-organized to have $NEW_ENTRIES_NB$ lists entries. Each entry is limited to $NEW_LISTS_MAX_SIZE$ elements. After a call to this function, all insertions into H will make H automatically growing if needed. The grow is needed each time an insertion makes an entry list to reach $NEW_LISTS_MAX_SIZE$ elements. In this case, H will be reorganized automatically by **`gdsl_hash_insert()`** (p. 96).

Note

Complexity: $O(|H|)$

Precondition

H must be a valid `gdsl_hash_t` & $NEW_ENTRIES_NB > \text{gdsl_hash_get_entries_number}(H)$ & $NEW_LISTS_MAX_SIZE > \text{gdsl_hash_get_lists_max_size}(H)$

Parameters

H	The hashtable to modify
$NEW_ENTRIES_NB$	
$NEW_LISTS_MAX_SIZE$	Generated on Tue Aug 21 2012 16:00:00 for gdsl by Doxygen

Returns

the modified hashtable H in case of success
 NULL in case of failure, or in case `NEW_ENTRIES_NB <= gdsi_hash_get_entries_number(H)` or in case `NEW_LISTS_MAX_SIZE <= gdsi_hash_get_lists_max_size(H)` in these cases, H is not modified

See also

`gdsi_hash_insert()` (p. 96)
`gdsi_hash_get_entries_number()` (p. 92)
`gdsi_hash_get_fill_factor()` (p. 95)
`gdsi_hash_get_longest_list_size()` (p. 93)
`gdsi_hash_get_lists_max_size()` (p. 93)

4.8.2.16 `gdsi_element_t gdsi_hash_search(const gdsi_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 `gdsi_hash_t`

Parameters

<i>H</i>	The hashtable to search the element in
<i>KEY</i>	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

`gdsi_hash_insert()` (p. 96)
`gdsi_hash_remove()` (p. 97)
`gdsi_hash_delete()` (p. 97)

4.8.2.17 `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. 100) stops and returns its last examined element.

Note

Complexity: $O(|H|)$

Precondition

H must be a valid `gdsl_hash_t` & MAP_F != NULL

Parameters

<i>H</i>	The hashtable to map
<i>MAP_F</i>	The map function.
<i>USER_DATA</i>	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.

4.8.2.18 `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. -
Additionalnal 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

<i>H</i>	The hashtable to write.
<i>WRITE_F</i>	The write function.
<i>OUTPUT_FILE</i>	The file where to write H's elements.
<i>USER_DATA</i>	User's datas passed to <i>WRITE_F</i> .

See also

gdsi_hash_write_xml() (p. 101)

gdsi_hash_dump() (p. 102)

4.8.2.19 void **gdsi_hash_write_xml**(const **gdsi_hash_t** *H*, **gdsi_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 **gdsi_hash_t** & *OUTPUT_FILE* != NULL

Parameters

<i>H</i>	The hashtable to write.
<i>WRITE_F</i>	The write function.
<i>OUTPUT_FILE</i>	The file where to write H's elements.
<i>USER_DATA</i>	User's datas passed to <i>WRITE_F</i> .

See also

gdsi_hash_write() (p. 100)

gdsi_hash_dump() (p. 102)

4.8.2.20 `void gdsI_hash_dump(const gdsI_hash_t H, gdsI_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 gdsI_hash_t & OUTPUT_FILE != NULL

Parameters

<i>H</i>	The hashtable to write
<i>WRITE_F</i>	The write function
<i>OUTPUT_FILE</i>	The file where to write H's elements
<i>USER_DATA</i>	User's datas passed to WRITE_F

See also

`gdsI_hash_write()` (p. 100)

`gdsI_hash_write_xml()` (p. 101)

4.9 Heap manipulation module

Typedefs

- typedef struct heap * **gdsi_heap_t**
GDSL heap type.

Functions

- **gdsi_heap_t gdsi_heap_alloc** (const char *NAME, **gdsi_alloc_func_t** ALLOC_F, **gdsi_free_func_t** FREE_F, **gdsi_compare_func_t** COMP_F)
Create a new heap.
- void **gdsi_heap_free** (**gdsi_heap_t** H)
Destroy a heap.
- void **gdsi_heap_flush** (**gdsi_heap_t** H)
Flush a heap.
- const char * **gdsi_heap_get_name** (const **gdsi_heap_t** H)
Get the name of a heap.
- **ulong gdsi_heap_get_size** (const **gdsi_heap_t** H)
Get the size of a heap.
- **gdsi_element_t gdsi_heap_get_top** (const **gdsi_heap_t** H)
Get the top of a heap.
- **bool gdsi_heap_is_empty** (const **gdsi_heap_t** H)
Check if a heap is empty.
- **gdsi_heap_t gdsi_heap_set_name** (**gdsi_heap_t** H, const char *NEW_NAME)
Set the name of a heap.
- **gdsi_element_t gdsi_heap_set_top** (**gdsi_heap_t** H, void *VALUE)
Substitute the top element of a heap by a lesser one.
- **gdsi_element_t gdsi_heap_insert** (**gdsi_heap_t** H, void *VALUE)
Insert an element into a heap (PUSH).
- **gdsi_element_t gdsi_heap_remove_top** (**gdsi_heap_t** H)
Remove the top element from a heap (POP).
- **gdsi_heap_t gdsi_heap_delete_top** (**gdsi_heap_t** H)
Delete the top element from a heap.
- **gdsi_element_t gdsi_heap_map_forward** (const **gdsi_heap_t** H, **gdsi_map_func_t** MAP_F, void *USER_DATA)
Parse a heap.
- void **gdsi_heap_write** (const **gdsi_heap_t** H, **gdsi_write_func_t** WRITE_F, FILE *OUTPUT_FILE, void *USER_DATA)
Write all the elements of a heap to a file.
- void **gdsi_heap_write_xml** (const **gdsi_heap_t** H, **gdsi_write_func_t** WRITE_F, FILE *OUTPUT_FILE, void *USER_DATA)
Write the content of a heap to a file into XML.

- void **gdsi_heap_dump** (const **gdsi_heap_t** H, **gdsi_write_func_t** WRITE_F, FILE *OUTPUT_FILE, void *USER_DATA)

Dump the internal structure of a heap to a file.

4.9.1 Typedef Documentation

4.9.1.1 typedef struct heap* gdsi_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 54 of file gdsi_heap.h.

4.9.2 Function Documentation

4.9.2.1 **gdsi_heap_t gdsi_heap_alloc** (const char * NAME, **gdsi_alloc_func_t** ALLOC_F, **gdsi_free_func_t** FREE_F, **gdsi_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

gdsI_heap_free() (p. 105)
gdsI_heap_flush() (p. 105)

4.9.2.2 void gdsI_heap_free (gdsI_heap_t H)

Destroy a heap.

Deallocate all the elements of the heap H by calling H's FREE_F function passed to **gdsI_heap_alloc()** (p. 104). The name of H is deallocated and H is deallocated itself too.

Note

Complexity: $O(|H|)$

Precondition

H must be a valid gdsI_heap_t

Parameters

<i>H</i>	The heap to destroy
----------	---------------------

See also

gdsI_heap_alloc() (p. 104)
gdsI_heap_flush() (p. 105)

4.9.2.3 void gdsI_heap_flush (gdsI_heap_t H)

Flush a heap.

Deallocate all the elements of the heap H by calling H's FREE_F function passed to **gdsI_heap_alloc()** (p. 104). H is not deallocated itself and H's name is not modified.

Note

Complexity: $O(|H|)$

Precondition

H must be a valid `gdsI_heap_t`

Parameters

<i>H</i>	The heap to flush
----------	-------------------

See also

`gdsI_heap_alloc()` (p. 104)

`gdsI_heap_free()` (p. 105)

4.9.2.4 `const char* gdsI_heap_get_name(const gdsI_heap_t H)`

Get the name of a heap.

Note

Complexity: $O(1)$

Precondition

H must be a valid `gdsI_heap_t`

Postcondition

The returned string MUST NOT be freed.

Parameters

<i>H</i>	The heap to get the name from
----------	-------------------------------

Returns

the name of the heap H.

See also

`gdsI_heap_set_name()` (p. 108)

4.9.2.5 `ulong gdsI_heap_get_size(const gdsI_heap_t H)`

Get the size of a heap.

Note

Complexity: $O(1)$

Precondition

H must be a valid `gdsl_heap_t`

Parameters

<i>H</i>	The heap to get the size from
----------	-------------------------------

Returns

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

4.9.2.6 `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

<i>H</i>	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. 109)

4.9.2.7 `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

<i>H</i>	The heap to check
----------	-------------------

Returns

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

4.9.2.8 `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

<i>H</i>	The heap to change the name
<i>NEW_NAME</i>	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. 106)

4.9.2.9 gdsI_element_t gdsI_heap_set_top(gdsI_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 gdsI_heap_t

Parameters

<i>H</i>	The heap to substitute the top element
<i>VALUE</i>	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

gdsI_heap_get_top() (p. 107)

4.9.2.10 gdsI_element_t gdsI_heap_insert(gdsI_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 gdsI_heap_t

Parameters

<i>H</i>	The heap to modify
<i>VALUE</i>	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

gdsI_heap_alloc() (p. 104)
gdsI_heap_remove()
gdsI_heap_delete()
gdsI_heap_get_size() (p. 106)

4.9.2.11 gdsI_element_t gdsI_heap_remove_top(gdsI_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 gdsI_heap_t

Parameters

<i>H</i>	The heap to modify
----------	--------------------

Returns

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

See also

gdsI_heap_insert() (p. 109)
gdsI_heap_delete_top() (p. 110)

4.9.2.12 gdsI_heap_t gdsI_heap_delete_top(gdsI_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 **gdsI_heap_alloc()** (p. 104), then H is returned.

Note

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

Precondition

H must be a valid `gdsi_heap_t`

Parameters

<i>H</i>	The heap to modify
----------	--------------------

Returns

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

See also

`gdsi_heap_insert()` (p. 109)
`gdsi_heap_remove_top()` (p. 110)

4.9.2.13 `gdsi_element_t gdsi_heap_map_forward(const gdsi_heap_t H,
gdsi_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 `gdsi_heap_map()` stops and returns its last examined element.

Note

Complexity: $O(|H|)$

Precondition

H must be a valid `gdsi_heap_t` & MAP_F != NULL

Parameters

<i>H</i>	The heap to map
<i>MAP_F</i>	The map function.
<i>USER_DATA</i>	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.

4.9.2.14 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

<i>H</i>	The heap to write.
<i>WRITE_F</i>	The write function.
<i>OUTPUT_FILE</i>	The file where to write <i>H</i> 's elements.
<i>USER_DATA</i>	User's datas passed to <i>WRITE_F</i> .

See also

gdsl_heap_write_xml() (p. 112)
gdsl_heap_dump() (p. 113)

4.9.2.15 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 `gdsI_heap_t` & `OUTPUT_FILE` != NULL

Parameters

<i>H</i>	The heap to write.
<i>WRITE_F</i>	The write function.
<i>OUTPUT_FILE</i>	The file where to write H's elements.
<i>USER_DATA</i>	User's datas passed to <i>WRITE_F</i> .

See also

`gdsI_heap_write()` (p. 112)
`gdsI_heap_dump()` (p. 113)

4.9.2.16 void **`gdsI_heap_dump`**(const `gdsI_heap_t` *H*, `gdsI_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 `gdsI_heap_t` & `OUTPUT_FILE` != NULL

Parameters

<i>H</i>	The heap to write
<i>WRITE_F</i>	The write function
<i>OUTPUT_FILE</i>	The file where to write H's elements
<i>USER_DATA</i>	User's datas passed to <i>WRITE_F</i>

See also

`gdsI_heap_write()` (p. 112)
`gdsI_heap_write_xml()` (p. 112)

4.10 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** 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 Typedef Documentation

4.10.1.1 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 53 of file `gdsl_interval_heap.h`.

4.10.2 Function Documentation

4.10.2.1 **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

<i>NAME</i>	The name of the new interval heap to create
<i>ALLOC_F</i>	Function to alloc element when inserting it in the interval heap
<i>FREE_F</i>	Function to free element when removing it from the interval heap
<i>COMP_F</i>	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

gdsI_interval_heap_free() (p. 116)

gdsI_interval_heap_flush() (p. 117)

4.10.2.2 void gdsI_interval_heap_free(gdsI_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 **gdsI_interval_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 gdsI_interval_heap_t

Parameters

<i>H</i>	The interval heap to destroy
----------	------------------------------

See also

gdsI_interval_heap_alloc() (p. 115)

gdsI_interval_heap_flush() (p. 117)

4.10.2.3 void gdsI_interval_heap_flush(gdsI_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 **gdsI_interval_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 `gdsI_interval_heap_t`

Parameters

<i>H</i>	The heap to flush
----------	-------------------

See also

gdsI_interval_heap_alloc() (p. 115)

gdsI_interval_heap_free() (p. 116)

4.10.2.4 const char* gdsI_interval_heap_get_name(const gdsI_interval_heap_t *H*)

Get the name of an interval heap.

Note

Complexity: $O(1)$

Precondition

H must be a valid `gdsI_interval_heap_t`

Postcondition

The returned string **MUST NOT** be freed.

Parameters

H	The interval heap to get the name from
-----	--

the name of the interval heap H .

gdsi_interval_heap_set_name() (p. 119)

Complexity: $O(1)$

H must be a valid `gdsi_interval_heap_t`

H	The interval heap to get the size from
-----	--

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

Complexity: $O(1)$

H must be a valid `gdsi_interval_heap_t`

Parameters

<i>H</i>	The interval heap to get the size from
<i>size</i>	The new maximum size

Returns

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

4.10.2.7 bool gdsI_interval_heap_is_empty(const gdsI_interval_heap_t H)

Check if an interval heap is empty.

Note

Complexity: $O(1)$

Precondition

H must be a valid gdsI_interval_heap_t

Parameters

<i>H</i>	The interval heap to check
----------	----------------------------

Returns

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

4.10.2.8 gdsI_interval_heap_t gdsI_interval_heap_set_name (gdsI_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 gdsI_interval_heap_t

Parameters

<i>H</i>	The interval heap to change the name
<i>NEW_NAME-E</i>	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. 117)

4.10.2.9 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

<i>H</i>	The interval heap to modify
<i>VALUE</i>	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. 115)
gdsl_interval_heap_remove()
gdsl_interval_heap_delete()
gdsl_interval_heap_get_size() (p. 118)

4.10.2.10 `gdsI_element_t gdsI_interval_heap_remove_max (gdsI_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 `gdsI_interval_heap_t`

Parameters

<i>H</i>	The interval heap to modify
----------	-----------------------------

Returns

the removed top element.

NULL if the interval heap is empty.

See also

`gdsI_interval_heap_insert()` (p. 120)

`gdsI_interval_heap_delete_max()` (p. 123)

4.10.2.11 `gdsI_element_t gdsI_interval_heap_remove_min (gdsI_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 `gdsI_interval_heap_t`

Parameters

<i>H</i>	The interval heap to modify
----------	-----------------------------

Returns

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

See also

gdsI_interval_heap_insert() (p. 120)
gdsI_interval_heap_delete_max() (p. 123)

**4.10.2.12 gdsI_element_t gdsI_interval_heap_get_min (const
gdsI_interval_heap_t *H*)**

Get the minimum element.

Note

Complexity: $O(1)$

Precondition

H must be a valid gdsI_interval_heap_t

Parameters

<i>H</i>	The interval heap to get the size from
----------	--

Returns

The smallest element in *H*

**4.10.2.13 gdsI_element_t gdsI_interval_heap_get_max (const
gdsI_interval_heap_t *H*)**

Get the maximum element.

Note

Complexity: $O(1)$

Precondition

H must be a valid gdsI_interval_heap_t

Parameters

<i>H</i>	The interval heap to get the size from
----------	--

Returns

The largest element in H

4.10.2.14 `gdsI_interval_heap_t gdsI_interval_heap_delete_min (gdsI_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 **`gdsI_interval_heap_alloc()`** (p. 115), then H is returned.

Note

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

Precondition

H must be a valid `gdsI_interval_heap_t`

Parameters

<i>H</i>	The interval heap to modify
----------	-----------------------------

Returns

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

See also

`gdsI_interval_heap_insert()` (p. 120)
`gdsI_interval_heap_remove_top()`

4.10.2.15 `gdsI_interval_heap_t gdsI_interval_heap_delete_max (gdsI_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 **`gdsI_interval_heap_alloc()`** (p. 115), then H is returned.

Note

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

Precondition

H must be a valid `gdsl_interval_heap_t`

Parameters

<i>H</i>	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. 120)
`gdsl_interval_heap_remove_top()`

**4.10.2.16 `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 `GD_SL_MAP_STOP` then `gdsl_interval_heap_map()` stops and returns its last examined element.

Note

Complexity: $O(|H|)$

Precondition

H must be a valid `gdsl_interval_heap_t` & `MAP_F` != NULL

Parameters

<i>H</i>	The interval heap to map
<i>MAP_F</i>	The map function.
<i>USER_DATA</i>	User's datas passed to <code>MAP_F</code>

Returns

the first element for which MAP_F returns GD_SL_MAP_STOP.
 NULL when the parsing is done.

4.10.2.17 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

<i>H</i>	The interval heap to write.
<i>WRITE_F</i>	The write function.
<i>OUTPUT_FILE</i>	The file where to write <i>H</i> 's elements.
<i>USER_DATA</i>	User's datas passed to <i>WRITE_F</i> .

See also

gdsl_interval_heap_write_xml() (p. 125)
gdsl_interval_heap_dump() (p. 126)

4.10.2.18 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 `gdsI_interval_heap_t` & `OUTPUT_FILE` != NULL

Parameters

<i>H</i>	The interval heap to write.
<i>WRITE_F</i>	The write function.
<i>OUTPUT_FILE</i>	The file where to write H's elements.
<i>USER_DATA</i>	User's datas passed to <i>WRITE_F</i> .

See also

`gdsI_interval_heap_write()` (p. 125)
`gdsI_interval_heap_dump()` (p. 126)

4.10.2.19 **`void gdsI_interval_heap_dump(const gdsI_interval_heap_t H,
gdsI_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`. Additional `USER_DATA` argument could be passed to `WRITE_F`.

Note

Complexity: $O(|H|)$

Precondition

H must be a valid `gdsI_interval_heap_t` & `OUTPUT_FILE` != NULL

Parameters

<i>H</i>	The interval heap to write
<i>WRITE_F</i>	The write function
<i>OUTPUT_FILE</i>	The file where to write H's elements
<i>USER_DATA</i>	User's datas passed to <i>WRITE_F</i>

See also

`gdsI_interval_heap_write()` (p. 125)
`gdsI_interval_heap_write_xml()` (p. 125)

4.11 Doubly-linked list manipulation module

Typedefs

- typedef struct _gdsl_list * **gdsl_list_t**
GDSDL doubly-linked list type.
- typedef struct _gdsl_list_cursor * **gdsl_list_cursor_t**
GDSDL 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.

- **gdslist_t gdslist_delete_tail** (gdslist_t L)
Delete the tail of a list.
- **gdslist_t gdslist_delete** (gdslist_t L, gdslist_compare_func_t COMP_F, const void *VALUE)
Delete a particular element from a list.
- **gdslist_element_t gdslist_search** (const gdslist_t L, gdslist_compare_func_t COMP_F, const void *VALUE)
Search for a particular element into a list.
- **gdslist_element_t gdslist_search_by_position** (const gdslist_t L, ulong POS)
Search for an element by its position in a list.
- **gdslist_element_t gdslist_search_max** (const gdslist_t L, gdslist_compare_func_t COMP_F)
Search for the greatest element of a list.
- **gdslist_element_t gdslist_search_min** (const gdslist_t L, gdslist_compare_func_t COMP_F)
Search for the lowest element of a list.
- **gdslist_t gdslist_sort** (gdslist_t L, gdslist_compare_func_t COMP_F)
Sort a list.
- **gdslist_element_t gdslist_map_forward** (const gdslist_t L, gdslist_map_func_t MAP_F, void *USER_DATA)
Parse a list from head to tail.
- **gdslist_element_t gdslist_map_backward** (const gdslist_t L, gdslist_map_func_t MAP_F, void *USER_DATA)
Parse a list from tail to head.
- **void gdslist_write** (const gdslist_t L, gdslist_write_func_t WRITE_F, FILE *OUTPUT_FILE, void *USER_DATA)
Write all the elements of a list to a file.
- **void gdslist_write_xml** (const gdslist_t L, gdslist_write_func_t WRITE_F, FILE *OUTPUT_FILE, void *USER_DATA)
Write the content of a list to a file into XML.
- **void gdslist_dump** (const gdslist_t L, gdslist_write_func_t WRITE_F, FILE *OUTPUT_FILE, void *USER_DATA)
Dump the internal structure of a list to a file.
- **gdslist_cursor_t gdslist_cursor_alloc** (const gdslist_t L)
Create a new list cursor.
- **void gdslist_cursor_free** (gdslist_cursor_t C)
Destroy a list cursor.
- **void gdslist_cursor_move_to_head** (gdslist_cursor_t C)
Put a cursor on the head of its list.
- **void gdslist_cursor_move_to_tail** (gdslist_cursor_t C)
Put a cursor on the tail of its list.
- **gdslist_element_t gdslist_cursor_move_to_value** (gdslist_cursor_t C, gdslist_compare_func_t COMP_F, void *VALUE)

Place a cursor on a particular element.

- **gdsI_element_t gdsI_list_cursor_move_to_position** (gdsI_list_cursor_t C, ulong POS)

Place a cursor on a element given by its position.

- void **gdsI_list_cursor_step_forward** (gdsI_list_cursor_t C)

Move a cursor one step forward of its list.

- void **gdsI_list_cursor_step_backward** (gdsI_list_cursor_t C)

Move a cursor one step backward of its list.

- **bool gdsI_list_cursor_is_on_head** (const gdsI_list_cursor_t C)

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

- **bool gdsI_list_cursor_is_on_tail** (const gdsI_list_cursor_t C)

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

- **bool gdsI_list_cursor_has_succ** (const gdsI_list_cursor_t C)

Check if a cursor has a successor.

- **bool gdsI_list_cursor_has_pred** (const gdsI_list_cursor_t C)

Check if a cursor has a predecessor.

- void **gdsI_list_cursor_set_content** (gdsI_list_cursor_t C, gdsI_element_t E)

Set the content of the cursor.

- **gdsI_element_t gdsI_list_cursor_get_content** (const gdsI_list_cursor_t C)

Get the content of a cursor.

- **gdsI_element_t gdsI_list_cursor_insert_after** (gdsI_list_cursor_t C, void *VALUE)

Insert a new element after a cursor.

- **gdsI_element_t gdsI_list_cursor_insert_before** (gdsI_list_cursor_t C, void *VALUE)

Insert a new element before a cursor.

- **gdsI_element_t gdsI_list_cursor_remove** (gdsI_list_cursor_t C)

Remove the element under a cursor.

- **gdsI_element_t gdsI_list_cursor_remove_after** (gdsI_list_cursor_t C)

Remove the element after a cursor.

- **gdsI_element_t gdsI_list_cursor_remove_before** (gdsI_list_cursor_t C)

Remove the element before a cursor.

- **gdsI_list_cursor_t gdsI_list_cursor_delete** (gdsI_list_cursor_t C)

Delete the element under a cursor.

- **gdsI_list_cursor_t gdsI_list_cursor_delete_after** (gdsI_list_cursor_t C)

Delete the element after a cursor.

- **gdsI_list_cursor_t gdsI_list_cursor_delete_before** (gdsI_list_cursor_t C)

Delete the element before the cursor of a list.

4.11.1 Typedef Documentation

4.11.1.1 `typedef struct _gdsl_list* gsdl_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 51 of file `gsdl_list.h`.

4.11.1.2 `typedef struct _gsdl_list_cursor* gsdl_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 59 of file `gsdl_list.h`.

4.11.2 Function Documentation

4.11.2.1 `gsdl_list_t gsdl_list_alloc(const char * NAME, gsdl_alloc_func_t ALLOC_F, gsdl_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

<i>NAME</i>	The name of the new list to create
<i>ALLOC_F</i>	Function to alloc element when inserting it in the list
<i>FREE_F</i>	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

gdsI_list_free() (p. 131)
gdsI_list_flush() (p. 131)

4.11.2.2 void gdsI_list_free(gdsI_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 **gdsI_list_alloc()** (p. 130).

Note

Complexity: $O(|L|)$

Precondition

L must be a valid gdsI_list_t

Parameters

L	The list to destroy
----------	---------------------

See also

gdsI_list_alloc() (p. 130)
gdsI_list_flush() (p. 131)

4.11.2.3 void gdsI_list_flush(gdsI_list_t L)

Flush a list.

Destroy all the elements of the list L by calling L's FREE_F function passed to **gdsI_list_alloc()** (p. 130). L is not deallocated itself and L's name is not modified.

Note

Complexity: $O(|L|)$

Precondition

L must be a valid `gdsI_list_t`

Parameters

<i>L</i>	The list to flush
----------	-------------------

See also

`gdsI_list_alloc()` (p. 130)

`gdsI_list_free()` (p. 131)

4.11.2.4 `const char* gdsI_list_get_name(const gdsI_list_t L)`

Get the name of a list.

Note

Complexity: $O(1)$

Precondition

L must be a valid `gdsI_list_t`

Postcondition

The returned string MUST NOT be freed.

Parameters

<i>L</i>	The list to get the name from
----------	-------------------------------

Returns

the name of the list L.

See also

`gdsI_list_set_name()` (p. 134)

4.11.2.5 `ulong gdsI_list_get_size(const gdsI_list_t L)`

Get the size of a list.

Note

Complexity: $O(1)$

Precondition

L must be a valid `gdsl_list_t`

Parameters

<code>L</code>	The list to get the size from
----------------	-------------------------------

Returns

the number of elements of the list L (noted $|L|$).

4.11.2.6 `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

<code>L</code>	The list to check
----------------	-------------------

Returns

TRUE if the list L is empty.
FALSE if the list L is not empty.

4.11.2.7 `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. 134)

4.11.2.8 `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. 133)

4.11.2.9 `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

<i>L</i>	The list to change the name
<i>NEW_NAME</i>	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. 132)

4.11.2.10 `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. 130). 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

<i>L</i>	The list to insert into
<i>VALUE</i>	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

gdsI_list_insert_tail() (p. 136)
gdsI_list_remove_head() (p. 136)
gdsI_list_remove_tail() (p. 137)
gdsI_list_remove() (p. 138)

4.11.2.11 gdsI_element_t gdsI_list_insert_tail(gdsI_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 **gdsI_list_alloc()** (p. 130). 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 gdsI_list_t

Parameters

<i>L</i>	The list to insert into
<i>VALUE</i>	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

gdsI_list_insert_head() (p. 135)
gdsI_list_remove_head() (p. 136)
gdsI_list_remove_tail() (p. 137)
gdsI_list_remove() (p. 138)

4.11.2.12 gdsI_element_t gdsI_list_remove_head(gdsI_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 `gds_l_list_t`

Parameters

<code>L</code>	The list to remove the head from
----------------	----------------------------------

Returns

the removed element in case of success.
NULL in case of L is empty.

See also

`gds_l_list_insert_head()` (p. 135)

`gds_l_list_insert_tail()` (p. 136)

`gds_l_list_remove_tail()` (p. 137)

`gds_l_list_remove()` (p. 138)

4.11.2.13 `gds_l_element_t gds_l_list_remove_tail(gds_l_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 `gds_l_list_t`

Parameters

<code>L</code>	The list to remove the tail from
----------------	----------------------------------

Returns

the removed element in case of success.
 NULL in case of L is empty.

See also

gdsI_list_insert_head() (p. 135)
gdsI_list_insert_tail() (p. 136)
gdsI_list_remove_head() (p. 136)
gdsI_list_remove() (p. 138)

4.11.2.14 **gdsI_element_t gdsI_list_remove(gdsI_list_t L, gdsI_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 gdsI_list_t & COMP_F != NULL

Parameters

<i>L</i>	The list to remove the element from
<i>COMP_F</i>	The comparison function used to find the element to remove
<i>VALUE</i>	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

gdsI_list_insert_head() (p. 135)
gdsI_list_insert_tail() (p. 136)
gdsI_list_remove_head() (p. 136)
gdsI_list_remove_tail() (p. 137)

4.11.2.15 `gdsl_list_t gsdl_list_delete_head(gsdl_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 `gsdl_list_alloc()` (p. 130).

Note

Complexity: $O(1)$

Precondition

L must be a valid `gsdl_list_t`

Parameters

<code>L</code>	The list to destroy the head from
----------------	-----------------------------------

Returns

the modified list L in case of success.

NULL if L is empty.

See also

`gsdl_list_alloc()` (p. 130)

`gsdl_list_destroy_tail()`

`gsdl_list_destroy()`

4.11.2.16 `gsdl_list_t gsdl_list_delete_tail(gsdl_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 `gsdl_list_alloc()` (p. 130).

Note

Complexity: $O(1)$

Precondition

L must be a valid `gsdl_list_t`

Parameters

<code>L</code>	The list to destroy the tail from
----------------	-----------------------------------

Returns

the modified list L in case of success.
 NULL if L is empty.

See also

gdsI_list_alloc() (p. 130)
 gdsI_list_destroy_head()
 gdsI_list_destroy()

4.11.2.17 **gdsI_list_t gdsI_list_delete(gdsI_list_t L, gdsI_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 **gdsI_list_alloc()** (p. 130).

Note

Complexity: $O(|L| / 2)$

Precondition

L must be a valid gdsI_list_t & COMP_F != NULL

Parameters

<i>L</i>	The list to destroy the element from
<i>COMP_F</i>	The comparison function used to find the element to destroy
<i>VALUE</i>	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

gdsI_list_alloc() (p. 130)
 gdsI_list_destroy_head()
 gdsI_list_destroy_tail()

4.11.2.18 **gdsI_element_t gdsI_list_search** (**const gdsI_list_t** *L*,
gdsI_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 **gdsI_list_t** & *COMP_F* != NULL

Parameters

<i>L</i>	The list to search the element in
<i>COMP_F</i>	The comparison function used to compare <i>L</i> 's element with <i>VALUE</i>
<i>VALUE</i>	The value to compare <i>L</i> 's element with

Returns

the first founded element *E* in case of success.
NULL in case the searched element *E* was not found.

See also

gdsI_list_search_by_position() (p. 141)
gdsI_list_search_max() (p. 142)
gdsI_list_search_min() (p. 143)

4.11.2.19 **gdsI_element_t gdsI_list_search_by_position** (**const gdsI_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 **gdsI_list_t** & $POS > 0$ & $POS \leq |L|$

Parameters

<i>L</i>	The list to search the element in
<i>POS</i>	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 $POS \leq 0$.

See also

gdsI_list_search() (p. 141)
gdsI_list_search_max() (p. 142)
gdsI_list_search_min() (p. 143)

4.11.220 **gdsI_element_t gdsI_list_search_max(const gdsI_list_t L, gdsI_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 gdsI_list_t & COMP_F != NULL

Parameters

<i>L</i>	The list to search the element in
<i>COMP_F</i>	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

gdsI_list_search() (p. 141)
gdsI_list_search_by_position() (p. 141)
gdsI_list_search_min() (p. 143)

**4.11.2.21 `gdsI_element_t gdsI_list_search_min (const gdsI_list_t L,
gdsI_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 gdsI_list_t & COMP_F != NULL

Parameters

<i>L</i>	The list to search the element in
<i>COMP_F</i>	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

`gdsI_list_search()` (p. 141)
`gdsI_list_search_by_position()` (p. 141)
`gdsI_list_search_max()` (p. 142)

4.11.2.22 `gdsI_list_t gdsI_list_sort (gdsI_list_t L, gdsI_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 gdsI_list_t & COMP_F != NULL & L must not contains elements that are equals

Parameters

<i>L</i>	The list to sort
<i>COMP_F</i>	The comparison function used to order L's elements

Returns

the sorted list *L*.

4.11.2.23 `gdsi_element_t gdsi_list_map_forward(const gdsi_list_t L,
gdsi_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 **`gdsi_list_map_forward()`** (p. 144) stops and returns its last examined element.

Note

Complexity: $O(|L|)$

Precondition

L must be a valid *gdsi_list_t* & *MAP_F* != NULL

Parameters

<i>L</i>	The list to parse
<i>MAP_F</i>	The map function to apply on each <i>L</i> 's element
<i>USER_DATA</i>	User's datas passed to <i>MAP_F</i>

Returns

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

See also

`gdsi_list_map_backward()` (p. 144)

4.11.2.24 `gdsi_element_t gdsi_list_map_backward(const gdsi_list_t L,
gdsi_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. 144) stops and returns its last examined element.

Note

Complexity: $O(|L|)$

Precondition

L must be a valid *gdsl_list_t* & *MAP_F* != NULL

Parameters

<i>L</i>	The list to parse
<i>MAP_F</i>	The map function to apply on each <i>L</i> 's element
<i>USER_DATA</i>	User's datas passed to <i>MAP_F</i>

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. 144)

4.11.2.25 `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. Additional *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

<i>L</i>	The list to write.
<i>WRITE_F</i>	The write function.
<i>OUTPUT_FILE</i>	The file where to write L's elements.
<i>USER_DATA</i>	User's datas passed to WRITE_F.

See also

gdsi_list_write_xml() (p. 146)

gdsi_list_dump() (p. 147)

4.11.2.26 `void gdsi_list_write_xml(const gdsi_list_t L, gdsi_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 gdsi_list_t & OUTPUT_FILE != NULL

Parameters

<i>L</i>	The list to write.
<i>WRITE_F</i>	The write function.
<i>OUTPUT_FILE</i>	The file where to write L's elements.
<i>USER_DATA</i>	User's datas passed to WRITE_F.

See also

gdsi_list_write() (p. 145)

gdsi_list_dump() (p. 147)

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

Dump the internal structure of a list to a file.

Dump the structure of the list `L` to `OUTPUT_FILE`. If `WRITE_F` \neq `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` \neq `NULL`

Parameters

<code>L</code>	The list to write.
<code>WRITE_F</code>	The write function.
<code>OUTPUT_FILE</code>	The file where to write <code>L</code> 's elements.
<code>USER_DATA</code>	User's datas passed to <code>WRITE_F</code> .

See also

`gdsl_list_write()` (p. 145)

`gdsl_list_write_xml()` (p. 146)

4.11.2.28 `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

<code>L</code>	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

gdsI_list_cursor_free() (p. 148)

4.11.2.29 void gdsI_list_cursor_free(gdsI_list_cursor_t C)

Destroy a list cursor.

Note

Complexity: $O(1)$

Precondition

C must be a valid gdsI_list_cursor_t.

Parameters

C	The list cursor to destroy.
---	-----------------------------

See also

gdsI_list_cursor_alloc() (p. 147)

4.11.2.30 void gdsI_list_cursor_move_to_head(gdsI_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 gdsI_list_cursor_t

Parameters

C	The cursor to use
---	-------------------

See also

gdsI_list_cursor_move_to_tail() (p. 149)

4.11.2.31 void **gdsI_list_cursor_move_to_tail**(**gdsI_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 **gdsI_list_cursor_t**

Parameters

<i>C</i>	The cursor to use
----------	-------------------

See also

gdsI_list_cursor_move_to_head() (p. 148)

4.11.2.32 **gdsI_element_t gdsI_list_cursor_move_to_value**(**gdsI_list_cursor_t** *C*,
gdsI_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 **gdsI_list_cursor_t** & *COMP_F* != NULL

Parameters

<i>C</i>	The cursor to put on the element <i>E</i>
<i>COMP_F</i>	The comparison function to search for <i>E</i>
<i>VALUE</i>	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. 150)

4.11.2.33 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 & $POS > 0$ & $POS \leq |L|$

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 $POS \leq 0$ or $POS > |L|$

See also

gdsl_list_cursor_move_to_value() (p. 149)

4.11.2.34 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 `gdsI_list_cursor_t`

Parameters

C	The cursor to use
---	-------------------

See also

`gdsI_list_cursor_step_backward()` (p. 151)

4.11.2.35 void `gdsI_list_cursor_step_backward(gdsI_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 `gdsI_list_cursor_t`

Parameters

C	The cursor to use
---	-------------------

See also

`gdsI_list_cursor_step_forward()` (p. 150)

4.11.2.36 bool `gdsI_list_cursor_is_on_head(const gdsI_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 list's head.

See also

`gdsl_list_cursor_is_on_tail()` (p. 152)

4.11.2.37 **`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 list's tail.
FALSE if C is not on its list's tail.

See also

`gdsl_list_cursor_is_on_head()` (p. 151)

4.11.2.38 **`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 `gdsI_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

`gdsI_list_cursor_has_pred()` (p. 153)

4.11.2.39 `bool gdsI_list_cursor_has_pred(const gdsI_list_cursor_t C)`

Check if a cursor has a predecessor.

Note

Complexity: $O(1)$

Precondition

C must be a valid `gdsI_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

`gdsI_list_cursor_has_succ()` (p. 152)

4.11.2.40 void `gdsI_list_cursor_set_content` (`gdsI_list_cursor_t` *C*, `gdsI_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, `gdsI_list_cursor_get_content()` (p. 154) could be used to get it in order to free it before.

Note

Complexity: $O(1)$

Precondition

C must be a valid `gdsI_list_cursor_t`

Parameters

<i>C</i>	The cursor in which the content must be modified.
<i>E</i>	The value used to modify <i>C</i> 's content.

See also

`gdsI_list_cursor_get_content()` (p. 154)

4.11.2.41 `gdsI_element_t` `gdsI_list_cursor_get_content` (const `gdsI_list_cursor_t` *C*)

Get the content of a cursor.

Note

Complexity: $O(1)$

Precondition

C must be a valid `gdsI_list_cursor_t`

Parameters

<i>C</i>	The cursor to get the content from.
----------	-------------------------------------

Returns

the element contained in the cursor *C*.

See also

gdsI_list_cursor_set_content() (p. 154)

4.11.2.42 **gdsI_element_t gdsI_list_cursor_insert_after(gdsI_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 **gdsI_list_alloc()** (p. 130). 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 **gdsI_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

gdsI_list_cursor_insert_before() (p. 155)

gdsI_list_cursor_remove_after() (p. 157)

gdsI_list_cursor_remove_before() (p. 157)

4.11.2.43 **gdsI_element_t gdsI_list_cursor_insert_before(gdsI_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 **gdsI_list_alloc()** (p. 130). 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. 155)
`gdsl_list_cursor_remove_after()` (p. 157)
`gdsl_list_cursor_remove_before()` (p. 157)

4.11.2.44 `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

gdsI_list_cursor_insert_after() (p. 155)
gdsI_list_cursor_insert_before() (p. 155)
gdsI_list_cursor_remove() (p. 156)
gdsI_list_cursor_remove_before() (p. 157)

4.11.2.45 gdsI_element_t gdsI_list_cursor_remove_after(gdsI_list_cursor_t C)

Removec the element after a cursor.

Note

Complexity: $O(1)$

Precondition

C must be a valid gdsI_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

gdsI_list_cursor_insert_after() (p. 155)
gdsI_list_cursor_insert_before() (p. 155)
gdsI_list_cursor_remove() (p. 156)
gdsI_list_cursor_remove_before() (p. 157)

4.11.2.46 gdsI_element_t gdsI_list_cursor_remove_before(gdsI_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. 155)
`gdsl_list_cursor_insert_before()` (p. 155)
`gdsl_list_cursor_remove()` (p. 156)
`gdsl_list_cursor_remove_after()` (p. 157)

4.11.2.47 `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. 130).

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. 159)
`gdsl_list_cursor_delete_after()` (p. 159)

4.11.2.48 `gdsI_list_cursor_t gdsI_list_cursor_delete_after(gdsI_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 **`gdsI_list_alloc()`** (p. 130).

Complexity: $O(1)$

Precondition

C must be a valid `gdsI_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

`gdsI_list_cursor_delete()` (p. 158)
`gdsI_list_cursor_delete_before()` (p. 159)

4.11.2.49 `gdsI_list_cursor_t gdsI_list_cursor_delete_before(gdsI_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 **`gdsI_list_alloc()`** (p. 130).

Note

Complexity: $O(1)$

Precondition

C must be a valid `gdsI_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

gdsI_list_cursor_delete() (p. 158)

gdsI_list_cursor_delete_after() (p. 159)

4.12 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.

4.12.1 Define Documentation

4.12.1.1 #define **GDSL_MAX**(X, Y) (X>Y?X:Y)

Give the greatest number of two numbers.

Note

Complexity: O(1)

Precondition

X & Y must be basic scalar C types

Parameters

X	First scalar variable
Y	Second scalar variable

Returns

X if X is greather than Y.
Y if Y is greather than X.

See also

GDSL_MIN() (p. 161)

Definition at line 56 of file gdsI_macros.h.

4.12.1.2 #define **GDSL_MIN**(X, Y) (X>Y?Y:X)

Give the lowest number of two numbers.

Note

Complexity: O(1)

Precondition

X & Y must be basic scalar C types

Parameters

X	First scalar variable
Y	Second scalar variable

Returns

Y if Y is lower than X.
X if X is lower than Y.

See also

GDSL_MAX() (p. 161)

Definition at line 73 of file gdsl_macros.h.

4.13 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_PERM_POSITION_LAST** = 2 }
- This type is for gdsl_perm_write_func_t.*

Functions

- **gdsl_perm_t gdsl_perm_alloc** (const char *NAME, const ulong N)
Create a new permutation.
- void **gdsl_perm_free** (gdsl_perm_t P)
Destroy a permutation.
- **gdsl_perm_t gdsl_perm_copy** (const gdsl_perm_t P)
Copy a permutation.
- const char * **gdsl_perm_get_name** (const gdsl_perm_t P)
Get the name of a permutation.
- ulong **gdsl_perm_get_size** (const gdsl_perm_t P)
Get the size of a permutation.
- ulong **gdsl_perm_get_element** (const gdsl_perm_t P, const ulong INDIX)
Get the (INDIX+1)-th element from a permutation.
- ulong * **gdsl_perm_get_elements_array** (const gdsl_perm_t P)
Get the array elements of a permutation.
- ulong **gdsl_perm_linear_inversions_count** (const gdsl_perm_t P)
Count the inversions number into a linear permutation.
- ulong **gdsl_perm_linear_cycles_count** (const gdsl_perm_t P)
Count the cycles number into a linear permutation.
- ulong **gdsl_perm_canonical_cycles_count** (const gdsl_perm_t P)
Count the cycles number into a canonical permutation.
- **gdsl_perm_t gdsl_perm_set_name** (gdsl_perm_t P, const char *NEW_NAME)
Set the name of a permutation.
- **gdsl_perm_t gdsl_perm_linear_next** (gdsl_perm_t P)

Get the next permutation from a linear permutation.

- **gdsI_perm_t gdsI_perm_linear_prev** (gdsI_perm_t P)

Get the previous permutation from a linear permutation.

- **gdsI_perm_t gdsI_perm_set_elements_array** (gdsI_perm_t P, const ulong *-ARRAY)

Initialize a permutation with an array of values.

- **gdsI_perm_t gdsI_perm_multiply** (gdsI_perm_t RESULT, const gdsI_perm_t ALPHA, const gdsI_perm_t BETA)

Multiply two permutations.

- **gdsI_perm_t gdsI_perm_linear_to_canonical** (gdsI_perm_t Q, const gdsI_perm_t P)

Convert a linear permutation to its canonical form.

- **gdsI_perm_t gdsI_perm_canonical_to_linear** (gdsI_perm_t Q, const gdsI_perm_t P)

Convert a canonical permutation to its linear form.

- **gdsI_perm_t gdsI_perm_inverse** (gdsI_perm_t P)

Inverse in place a permutation.

- **gdsI_perm_t gdsI_perm_reverse** (gdsI_perm_t P)

Reverse in place a permutation.

- **gdsI_perm_t gdsI_perm_randomize** (gdsI_perm_t P)

Randomize a permutation.

- **gdsI_element_t * gdsI_perm_apply_on_array** (gdsI_element_t *V, const gdsI_perm_t P)

Apply a permutation on to a vector.

- void **gdsI_perm_write** (const gdsI_perm_t P, const gdsI_write_func_t WRITE_F, FILE *OUTPUT_FILE, void *USER_DATA)

Write the elements of a permutation to a file.

- void **gdsI_perm_write_xml** (const gdsI_perm_t P, const gdsI_write_func_t WRITE_F, FILE *OUTPUT_FILE, void *USER_DATA)

Write the elements of a permutation to a file into XML.

- void **gdsI_perm_dump** (const gdsI_perm_t P, const gdsI_write_func_t WRITE_F, FILE *OUTPUT_FILE, void *USER_DATA)

Dump the internal structure of a permutation to a file.

4.13.1 Typedef Documentation

4.13.1.1 typedef struct gdsI_perm* gdsI_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 50 of file gdsI_perm.h.

4.13.1.2 `typedef void(* gdsI_perm_write_func_t)(ulong E, FILE *OUTPUT_FILE,
gdsI_location_t POSITION, void *USER_DATA)`

GDSL permutation write function type.

Parameters

<i>E</i>	The permutation element to write
<i>OUTPUT_FILE</i>	The file where to write E
<i>POSITION</i>	is an or-ed combination of <code>gdsI_perm_position_t</code> values to indicate where E is located into the <code>gdsI_perm_t</code> mapped.
<i>USER_DATA</i>	User's datas

Definition at line 74 of file `gdsI_perm.h`.

4.13.1.3 `typedef struct gdsI_perm_data* gdsI_perm_data_t`

Definition at line 80 of file `gdsI_perm.h`.

4.13.2 Enumeration Type Documentation

4.13.2.1 `enum gdsI_perm_position_t`

This type is for `gdsI_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 55 of file `gdsI_perm.h`.

4.13.3 Function Documentation

4.13.3.1 `gdsI_perm_t gdsI_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

<i>N</i>	The number of elements of the permutation to create.
<i>NAME</i>	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

gdsI_perm_free() (p. 166)
gdsI_perm_copy() (p. 166)

4.13.3.2 void gdsI_perm_free (gdsI_perm_t P)

Destroy a permutation.

Deallocate the permutation P.

Note

Complexity: $O(|P|)$

Precondition

P must be a valid gdsI_perm_t

Parameters

<i>P</i>	The permutation to destroy
----------	----------------------------

See also

gdsI_perm_alloc() (p. 165)
gdsI_perm_copy() (p. 166)

4.13.3.3 gdsI_perm_t gdsI_perm_copy (const gdsI_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 `gdsI_perm_t`.

Postcondition

The returned permutation must be deallocated with `gdsI_perm_free`.

Parameters

<i>P</i>	The permutation to copy.
----------	--------------------------

Returns

a copy of P in case of success.
NULL in case of insufficient memory.

See also

`gdsI_perm_alloc` (p. 165)
`gdsI_perm_free` (p. 166)

4.13.3.4 `const char* gdsI_perm_get_name(const gdsI_perm_t P)`

Get the name of a permutation.

Note

Complexity: $O(1)$

Precondition

P must be a valid `gdsI_perm_t`

Postcondition

The returned string **MUST NOT** be freed.

Parameters

<i>P</i>	The permutation to get the name from
----------	--------------------------------------

Returns

the name of the permutation P .

See also

gdsI_perm_set_name() (p. 171)

4.13.3.5 `ulong gdsI_perm_get_size(const gdsI_perm_t P)`

Get the size of a permutation.

Note

Complexity: $O(1)$

Precondition

P must be a valid `gdsI_perm_t`

Parameters

P	The permutation to get the size from.
-----	---------------------------------------

Returns

the number of elements of P (noted $|P|$).

See also

gdsI_perm_get_element() (p. 168)

gdsI_perm_get_elements_array() (p. 169)

4.13.3.6 `ulong gdsI_perm_get_element(const gdsI_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 `gdsI_perm_t` & $0 \leq INDIX < |P|$

Parameters

<i>P</i>	The permutation to use.
<i>INDIX</i>	The index of the value to get.

Returns

the value at the *INDIX*-th position in the permutation *P*.

See also

gdsI_perm_get_size() (p. 168)

gdsI_perm_get_elements_array() (p. 169)

4.13.3.7 `ulong* gdsI_perm_get_elements_array(const gdsI_perm_t P)`

Get the array elements of a permutation.

Note

Complexity: $O(1)$

Precondition

P must be a valid `gdsI_perm_t`

Parameters

<i>P</i>	The permutation to get data from.
----------	-----------------------------------

Returns

the values array of the permutation *P*.

See also

gdsI_perm_get_element() (p. 168)

gdsI_perm_set_elements_array() (p. 173)

4.13.3.8 `ulong gdsI_perm_linear_inversions_count(const gdsI_perm_t P)`

Count the inversions number into a linear permutation.

Note

Complexity: $O(|P|)$

Precondition

P must be a valid linear `gdsI_perm_t`

Parameters

<i>P</i>	The linear permutation to use.
----------	--------------------------------

Returns

the number of inversions into the linear permutation P.

4.13.3.9 `ulong gdsI_perm_linear_cycles_count(const gdsI_perm_t P)`

Count the cycles number into a linear permutation.

Note

Complexity: $O(|P|)$

Precondition

P must be a valid linear `gdsI_perm_t`

Parameters

<i>P</i>	The linear permutation to use.
----------	--------------------------------

Returns

the number of cycles into the linear permutation P.

See also

`gdsI_perm_canonical_cycles_count()` (p. 170)

4.13.3.10 `ulong gdsI_perm_canonical_cycles_count(const gdsI_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

<i>P</i>	The canonical permutation to use.
----------	-----------------------------------

Returns

the number of cycles into the canonical permutation P.

See also

`gdsl_perm_linear_cycles_count()` (p. 170)

4.13.3.11 `gdsl_perm_t gsdl_perm_set_name (gsdl_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

<i>P</i>	The permutation to change the name
<i>NEW_NAME</i>	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. 167)

4.13.3.12 `gdsI_perm_t gdsI_perm_linear_next(gdsI_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 `gdsI_perm_t` & $|P| > 1$

Parameters

<i>P</i>	The linear permutation to modify
----------	----------------------------------

Returns

the next permutation after the permutation P.
NULL if P is already the last permutation.

See also

`gdsI_perm_linear_prev()` (p. 172)

4.13.3.13 `gdsI_perm_t gdsI_perm_linear_prev(gdsI_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 `gdsI_perm_t` & $|P| \geq 2$

Parameters

<i>P</i>	The linear permutation to modify
----------	----------------------------------

Returns

the previous permutation before the permutation P.
 NULL if P is already the first permutation.

See also

gdsI_perm_linear_next() (p. 172)

**4.13.3.14 gdsI_perm_t gdsI_perm_set_elements_array(gdsI_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 gdsI_perm_t & $V \neq \text{NULL}$ & $|V| == |P|$

Parameters

<i>P</i>	The permutation to initialize
<i>ARRAY</i>	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

gdsI_perm_get_elements_array() (p. 169)

**4.13.3.15 gdsI_perm_t gdsI_perm_multiply (gdsI_perm_t RESULT, const
 gdsI_perm_t ALPHA, const gdsI_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 `gdsI_perm_t` & $|RESULT| == |ALPHA| == |BETA|$

Parameters

<i>RESULT</i>	The result of the product ALPHA x BETA
<i>ALPHA</i>	The first permutation used in the product
<i>BETA</i>	The second permutation used in the product

Returns

RESULT, the result of the multiplication ALPHA x BETA.

4.13.3.16 `gdsI_perm_t gdsI_perm_linear_to_canonical(gdsI_perm_t Q, const gdsI_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 `gdsI_perm_t` & $|P| == |Q|$ & $P \neq Q$

Parameters

<i>Q</i>	The canonical form of P
<i>P</i>	The linear permutation used to compute its canonical form into Q

Returns

the canonical form Q of the permutation P.

See also

`gdsI_perm_canonical_to_linear()` (p. 175)

4.13.3.17 `gdsi_perm_t gdsi_perm_canonical_to_linear(gdsi_perm_t Q, const gdsi_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 `gdsi_perm_t` & $|P| == |Q|$ & $P \neq Q$

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

`gdsi_perm_linear_to_canonical()` (p. 174)

4.13.3.18 `gdsi_perm_t gdsi_perm_inverse(gdsi_perm_t P)`

Inverse in place a permutation.

Note

Complexity: $O(|P|)$

Precondition

P must be a valid `gdsi_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

gdsI_perm_reverse() (p. 176)

4.13.3.19 gdsI_perm_t gdsI_perm_reverse(gdsI_perm_t P)

Reverse in place a permutation.

Note

Complexity: $O(|P| / 2)$

Precondition

P must be a valid gdsI_perm_t

Parameters

P	The permutation to reverse
-----	----------------------------

Returns

the mirror image of the permutation P

See also

gdsI_perm_inverse() (p. 175)

4.13.3.20 gdsI_perm_t gdsI_perm_randomize(gdsI_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 gdsI_perm_t

Parameters

P	The permutation to randomize
-----	------------------------------

Returns

the mirror image $\sim P$ of the permutation of P in case of success.
 NULL in case of insufficient memory.

4.13.3.21 `gdsl_element_t* gdsi_perm_apply_on_array(gdsi_element_t* V, const gdsi_perm_t P)`

Apply a permutation on to a vector.

Note

Complexity: $O(|P|)$

Precondition

P must be a valid `gdsi_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.

4.13.3.22 `void gdsi_perm_write(const gdsi_perm_t P, const gdsi_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 permutation P to `OUTPUT_FILE`, using `WRITE_F` function.
 Additional `USER_DATA` argument could be passed to `WRITE_F`.

Note

Complexity: $O(|P|)$

Precondition

P must be a valid `gdsi_perm_t` & `WRITE_F` != NULL & `OUTPUT_FILE` != NULL

Parameters

<i>P</i>	The permutation to write.
<i>WRITE_F</i>	The write function.
<i>OUTPUT_FILE</i>	The file where to write P's elements.
<i>USER_DATA</i>	User's datas passed to WRITE_F.

See also

gdsl_perm_write_xml() (p. 178)

gdsl_perm_dump() (p. 179)

4.13.3.23 `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 WRITE_F != NULL, then uses WRITE_F function to write P's elements to OUTPUT_FILE. Additionnal USER_DATA argument could be passed to WRITE_F.

Note

Complexity: $O(|P|)$

Precondition

P must be a valid gdsl_perm_t & OUTPUT_FILE != NULL

Parameters

<i>P</i>	The permutation to write.
<i>WRITE_F</i>	The write function.
<i>OUTPUT_FILE</i>	The file where to write P's elements.
<i>USER_DATA</i>	User's datas passed to WRITE_F.

See also

gdsl_perm_write() (p. 177)

gdsl_perm_dump() (p. 179)

4.13.3.24 `void gdsi_perm_dump (const gdsi_perm_t P, const gdsi_write_func_t
WRITE_F, FILE * OUTPUT_FILE, void * USER_DATA)`

Dump the internal structure of a permutation to a file.

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

Note

Complexity: $O(|P|)$

Precondition

P must be a valid gdsi_perm_t & OUTPUT_FILE != NULL

Parameters

<i>P</i>	The permutation to dump.
<i>WRITE_F</i>	The write function.
<i>OUTPUT_FILE</i>	The file where to write P's elements.
<i>USER_DATA</i>	User's datas passed to WRITE_F.

See also

`gdsi_perm_write()` (p. 177)

`gdsi_perm_write_xml()` (p. 178)

4.14 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** 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)
Get the 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 **gdsI_queue_write** (const **gdsI_queue_t** Q, **gdsI_write_func_t** WRITE_F, FILE *OUTPUT_FILE, void *USER_DATA)

Write all the elements of a queue to a file.

- void **gdsI_queue_write_xml** (const **gdsI_queue_t** Q, **gdsI_write_func_t** WRITE_F, FILE *OUTPUT_FILE, void *USER_DATA)

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

- void **gdsI_queue_dump** (const **gdsI_queue_t** Q, **gdsI_write_func_t** WRITE_F, FILE *OUTPUT_FILE, void *USER_DATA)

Dump the internal structure of a queue to a file.

4.14.1 Typedef Documentation

4.14.1.1 typedef struct _gdsI_queue* gdsI_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 54 of file gdsI_queue.h.

4.14.2 Function Documentation

4.14.2.1 gdsI_queue_t gdsI_queue_alloc (const char * NAME, gdsI_alloc_func_t ALLOC_F, gdsI_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

<i>NAME</i>	The name of the new queue to create
<i>ALLOC_F</i>	Function to alloc element when inserting it in a queue
<i>FREE_F</i>	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

gdsI_queue_free() (p. 182)
gdsI_queue_flush() (p. 182)

4.14.2.2 void gdsI_queue_free(gdsI_queue_t Q)

Destroy a queue.

Deallocate all the elements of the queue Q by calling Q's FREE_F function passed to **gdsI_queue_alloc()** (p. 181). The name of Q is deallocated and Q is deallocated itself too.

Note

Complexity: $O(|Q|)$

Precondition

Q must be a valid gdsI_queue_t

Parameters

Q	The queue to destroy
---	----------------------

See also

gdsI_queue_alloc() (p. 181)
gdsI_queue_flush() (p. 182)

4.14.2.3 void gdsI_queue_flush(gdsI_queue_t Q)

Flush a queue.

Deallocate all the elements of the queue Q by calling Q's FREE_F function passed to gdsI_queue_allocc(). Q is not deallocated itself and Q's name is not modified.

Note

Complexity: $O(|Q|)$

Precondition

Q must be a valid `gdsI_queue_t`

Parameters

Q	The queue to flush
---	--------------------

See also

`gdsI_queue_alloc()` (p. 181)

`gdsI_queue_free()` (p. 182)

4.14.2.4 `const char* gdsI_queue_get_name(const gdsI_queue_t Q)`

Gets the name of a queue.

Note

Complexity: $O(1)$

Precondition

Q must be a valid `gdsI_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

`gdsI_queue_set_name()` (p. 186)

4.14.2.5 `ulong gdsI_queue_get_size(const gdsI_queue_t Q)`

Get the size of a queue.

Note

Complexity: $O(1)$

Precondition

Q must be a valid `gdsI_queue_t`

Parameters

Q	The queue to get the size from
---	--------------------------------

Returns

the number of elements of Q (noted $|Q|$).

4.14.2.6 `bool gdsI_queue_is_empty(const gdsI_queue_t Q)`

Check if a queue is empty.

Note

Complexity: $O(1)$

Precondition

Q must be a valid `gdsI_queue_t`

Parameters

Q	The queue to check
---	--------------------

Returns

TRUE if the queue Q is empty.
FALSE if the queue Q is not empty.

4.14.2.7 `gdsI_element_t gdsI_queue_get_head(const gdsI_queue_t Q)`

Get the head of a queue.

Note

Complexity: $O(1)$

Precondition

Q must be a valid `gdsI_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

`gdsI_queue_get_tail()` (p. 185)

4.14.2.8 `gdsI_element_t gdsI_queue_get_tail(const gdsI_queue_t Q)`

Get the tail of a queue.

Note

Complexity: $O(1)$

Precondition

Q must be a valid `gdsI_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

`gdsI_queue_get_head()` (p. 184)

4.14.2.9 `gdsl_queue_t` `gsdl_queue_set_name`(`gsdl_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 `gsdl_queue_t`

Parameters

<i>Q</i>	The queue to change the name
<i>NEW_NAME</i>	The new name of <i>Q</i>

Returns

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

See also

`gsdl_queue_get_name()` (p. 183)

4.14.2.10 `gsdl_element_t` `gsdl_queue_insert`(`gsdl_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 `gsdl_queue_alloc()` (p. 181). 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 `gsdl_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

gdsI_queue_remove() (p. 187)

4.14.2.11 gdsI_element_t gdsI_queue_remove(gdsI_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 gdsI_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

gdsI_queue_insert() (p. 186)

4.14.2.12 gdsI_element_t gdsI_queue_search(const gdsI_queue_t Q, gdsI_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 `gdsI_queue_t` & `COMP_F` != NULL

Parameters

Q	The queue to search the element in
<i>COMP_F</i>	The comparison function used to compare Q 's element with <i>VALUE</i>
<i>VALUE</i>	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

`gdsI_queue_search_by_position` (p. 188)

4.14.2.13 `gdsI_element_t gdsI_queue_search_by_position(const gdsI_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 `gdsI_queue_t` & $POS > 0$ & $POS \leq |Q|$

Parameters

Q	The queue to search the element in
<i>POS</i>	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 $POS \leq 0$.

See also

gdsl_queue_search() (p. 187)

4.14.2.14 `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. 189) stops and returns its last examined 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. 189)

4.14.2.15 `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. 189) stops and returns its last examined element.

Note

Complexity: $O(|Q|)$

Precondition

Q must be a valid `gdsl_queue_t` & $MAP_F \neq \text{NULL}$

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 <code>GDSL_MAP_STOP</code> . Returns <code>NULL</code> when the parsing is done.

See also

`gdsl_queue_map_forward()` (p. 189)

4.14.2.16 `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. -
Additional `USER_DATA` argument could be passed to `WRITE_F`.

Note

Complexity: $O(|Q|)$

Precondition

Q must be a valid `gdsl_queue_t` & `OUTPUT_FILE` $\neq \text{NULL}$ & `WRITE_F` $\neq \text{NULL}$

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 <code>WRITE_F</code> .

See also

gdsI_queue_write_xml() (p. 191)

gdsI_queue_dump() (p. 191)

4.14.2.17 void **gdsI_queue_write_xml**(const **gdsI_queue_t** *Q*, **gdsI_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 **gdsI_queue_t** & *OUTPUT_FILE* != NULL

Parameters

<i>Q</i>	The queue to write.
<i>WRITE_F</i>	The write function.
<i>OUTPUT_FILE</i>	The file where to write <i>Q</i> 's elements.
<i>USER_DATA</i>	User's datas passed to <i>WRITE_F</i> .

See also

gdsI_queue_write() (p. 190)

gdsI_queue_dump() (p. 191)

4.14.2.18 void **gdsI_queue_dump**(const **gdsI_queue_t** *Q*, **gdsI_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

<i>Q</i>	The queue to write.
<i>WRITE_F</i>	The write function.
<i>OUTPUT_FILE</i>	The file where to write Q's elements.
<i>USER_DATA</i>	User's datas passed to <i>WRITE_F</i> .

See also

`gdsl_queue_write()` (p. 190)

`gdsl_queue_write_xml()` (p. 191)

4.15 Red-black tree manipulation module

Typedefs

- typedef struct gdsi_rbtrees * **gdsi_rbtrees_t**

Functions

- **gdsi_rbtrees_t gdsi_rbtrees_alloc** (const char *NAME, **gdsi_alloc_func_t** ALL-OC_F, **gdsi_free_func_t** FREE_F, **gdsi_compare_func_t** COMP_F)
Create a new red-black tree.
- void **gdsi_rbtrees_free** (**gdsi_rbtrees_t** T)
Destroy a red-black tree.
- void **gdsi_rbtrees_flush** (**gdsi_rbtrees_t** T)
Flush a red-black tree.
- char * **gdsi_rbtrees_get_name** (const **gdsi_rbtrees_t** T)
Get the name of a red-black tree.
- bool **gdsi_rbtrees_is_empty** (const **gdsi_rbtrees_t** T)
Check if a red-black tree is empty.
- **gdsi_element_t** **gdsi_rbtrees_get_root** (const **gdsi_rbtrees_t** T)
Get the root of a red-black tree.
- **ulong** **gdsi_rbtrees_get_size** (const **gdsi_rbtrees_t** T)
Get the size of a red-black tree.
- **ulong** **gdsi_rbtrees_height** (const **gdsi_rbtrees_t** T)
Get the height of a red-black tree.
- **gdsi_rbtrees_t** **gdsi_rbtrees_set_name** (**gdsi_rbtrees_t** T, const char *NEW_NAME)
Set the name of a red-black tree.
- **gdsi_element_t** **gdsi_rbtrees_insert** (**gdsi_rbtrees_t** T, void *VALUE, int *RESULT)
Insert an element into a red-black tree if it's not found or return it.
- **gdsi_element_t** **gdsi_rbtrees_remove** (**gdsi_rbtrees_t** T, void *VALUE)
Remove an element from a red-black tree.
- **gdsi_rbtrees_t** **gdsi_rbtrees_delete** (**gdsi_rbtrees_t** T, void *VALUE)
Delete an element from a red-black tree.
- **gdsi_element_t** **gdsi_rbtrees_search** (const **gdsi_rbtrees_t** T, **gdsi_compare_func_t** COMP_F, void *VALUE)
Search for a particular element into a red-black tree.
- **gdsi_element_t** **gdsi_rbtrees_map_prefix** (const **gdsi_rbtrees_t** T, **gdsi_map_func_t** MAP_F, void *USER_DATA)
Parse a red-black tree in prefixed order.
- **gdsi_element_t** **gdsi_rbtrees_map_infix** (const **gdsi_rbtrees_t** T, **gdsi_map_func_t** MAP_F, void *USER_DATA)
Parse a red-black tree in infix order.

- **gdsl_element_t gsdl_rbtrees_map_postfix** (const **gsdl_rbtrees_t** T, **gsdl_map_func_t** MAP_F, void *USER_DATA)

Parse a red-black tree in postfix order.

- void **gsdl_rbtrees_write** (const **gsdl_rbtrees_t** T, **gsdl_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 **gsdl_rbtrees_write_xml** (const **gsdl_rbtrees_t** T, **gsdl_write_func_t** WRITE_F, FILE *OUTPUT_FILE, void *USER_DATA)

Write the content of a red-black tree to a file into XML.

- void **gsdl_rbtrees_dump** (const **gsdl_rbtrees_t** T, **gsdl_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 Typedef Documentation

4.15.1.1 typedef struct gsdl_rbtrees* gsdl_rbtrees_t

GDSL red-black tree type.

This type is voluntary opaque. Variables of this kind could not be directly used, but by the functions of this module.

Definition at line 52 of file gsdl_rbtrees.h.

4.15.2 Function Documentation

4.15.2.1 **gsdl_rbtrees_t gsdl_rbtrees_alloc** (const char * NAME, **gsdl_alloc_func_t** ALLOC_F, **gsdl_free_func_t** FREE_F, **gsdl_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

<i>NAME</i>	The name of the new red-black tree to create
<i>ALLOC_F</i>	Function to alloc element when inserting it in a r-b tree
<i>FREE_F</i>	Function to free element when removing it from a r-b tree
<i>COMP_F</i>	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

gdsI_rbtree_free() (p. 195)
gdsI_rbtree_flush() (p. 196)

4.15.2.2 void gdsI_rbtree_free (gdsI_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 **gdsI_rbtree_alloc()** (p. 194). The name of T is deallocated and T is deallocated itself too.

Note

Complexity: $O(|T|)$

Precondition

T must be a valid gdsI_rbtree_t

Parameters

<i>T</i>	The red-black tree to deallocate
----------	----------------------------------

See also

gdsI_rbtree_alloc() (p. 194)
gdsI_rbtree_flush() (p. 196)

4.15.2.3 void `gdsI_rbtree_flush(gdsI_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 **`gdsI_rbtree_alloc()`** (p. 194). 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 `gdsI_rbtree_t`

See also

`gdsI_rbtree_alloc()` (p. 194)

`gdsI_rbtree_free()` (p. 195)

4.15.2.4 char* `gdsI_rbtree_get_name(const gdsI_rbtree_t T)`

Get the name of a red-black tree.

Note

Complexity: $O(1)$

Precondition

`T` must be a valid `gdsI_rbtree_t`

Postcondition

The returned string MUST NOT be freed.

Parameters

<code>T</code>	The red-black tree to get the name from
----------------	---

Returns

the name of the red-black tree `T`.

See also

`gdsI_rbtreeset_name()` (p. 199)

4.15.2.5 `bool gdsI_rbtrees_is_empty(const gdsI_rbtrees_t T)`

Check if a red-black tree is empty.

Note

Complexity: $O(1)$

Precondition

T must be a valid `gdsI_rbtrees_t`

Parameters

<code>T</code>	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.

4.15.2.6 `gdsI_element_t gdsI_rbtrees_get_root(const gdsI_rbtrees_t T)`

Get the root of a red-black tree.

Note

Complexity: $O(1)$

Precondition

T must be a valid `gdsI_rbtrees_t`

Parameters

<code>T</code>	The red-black tree to get the root element from
----------------	---

Returns

the element at the root of the red-black tree T.

4.15.2.7 `ulong gdsi_rbtrees_get_size(const gdsi_rbtrees_t T)`

Get the size of a red-black tree.

Note

Complexity: $O(1)$

Precondition

T must be a valid `gdsi_rbtrees_t`

Parameters

<code>T</code>	The red-black tree to get the size from
----------------	---

Returns

the size of the red-black tree T (noted $|T|$).

See also

`gdsi_rbtrees_get_height()`

4.15.2.8 `ulong gdsi_rbtrees_height(const gdsi_rbtrees_t T)`

Get the height of a red-black tree.

Note

Complexity: $O(|T|)$

Precondition

T must be a valid `gdsi_rbtrees_t`

Parameters

<code>T</code>	The red-black tree to compute the height from
----------------	---

Returns

the height of the red-black tree T (noted $h(T)$).

See also

gdsI_rbtree_get_size() (p. 198)

4.15.2.9 **gdsI_rbtree_t gdsI_rbtree_set_name (gdsI_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 gdsI_rbtree_t

Parameters

<i>T</i>	The red-black tree to change the name
<i>NEW_NAME</i>	The new name of <i>T</i>

Returns

the modified red-black tree in case of success.
NULL in case of insufficient memory.

See also

gdsI_rbtree_get_name() (p. 196)

4.15.2.10 **gdsI_element_t gdsI_rbtree_insert (gdsI_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 gdsI_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 gdsI_rbtree_alloc and is inserted and then returned.

Note

Complexity: $O(\log(|T|))$

Precondition

T must be a valid `gdsl_rbtrees_t` & `RESULT` != NULL

Parameters

<i>T</i>	The red-black tree to modify
<i>VALUE</i>	The value used to make the new element to insert into T
<i>RESULT</i>	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_rbtrees_remove()` (p. 200)
`gdsl_rbtrees_delete()` (p. 201)

4.15.2.11 `gdsl_element_t gdsl_rbtrees_remove (gdsl_rbtrees_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_rbtrees_alloc()`** (p. 194). If E is found, it is removed from T and then returned.

Note

Complexity: $O(\log(|T|))$

Precondition

T must be a valid `gdsl_rbtrees_t`

Parameters

<i>T</i>	The red-black tree to modify
<i>VALUE</i>	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

gdsI_rbtree_insert() (p. 199)

gdsI_rbtree_delete() (p. 201)

4.15.2.12 **gdsI_rbtree_t gdsI_rbtree_delete(gdsI_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 **gdsI_rbtree_alloc()** (p. 194). If E is found, it is removed from T and E is deallocated using T's FREE_F function passed to **gdsI_rbtree_alloc()** (p. 194), then T is returned.

Note

Complexity: $O(\log(|T|))$

Precondition

T must be a valid gdsI_rbtree_t

Parameters

<i>T</i>	The red-black tree to remove an element from
<i>VALUE</i>	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

gdsI_rbtree_insert() (p. 199)

gdsI_rbtree_remove() (p. 200)

4.15.2.13 **gdsI_element_t gdsI_rbtree_search(const gdsI_rbtree_t T, gdsI_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 **gdsI_rbtree_alloc()** (p. 194) is used.

Note

Complexity: $O(\log(|T|))$

Precondition

T must be a valid `gdsi_rbtrees_t`

Parameters

<i>T</i>	The red-black tree to use.
<i>COMP_F</i>	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)
<i>VALUE</i>	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

`gdsi_rbtrees_insert()` (p. 199)
`gdsi_rbtrees_remove()` (p. 200)
`gdsi_rbtrees_delete()` (p. 201)

4.15.2.14 `gdsi_element_t gdsi_rbtrees_map_prefix (const gdsi_rbtrees_t T, gdsi_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 GDSI_MAP_STOP, then **`gdsi_rbtrees_map_prefix()`** (p. 202) stops and returns its last examined element.

Note

Complexity: $O(|T|)$

Precondition

T must be a valid `gdsi_rbtrees_t` & MAP_F != NULL

Parameters

<i>T</i>	The red-black tree to map.
<i>MAP_F</i>	The map function.
<i>USER_DATA</i>	User's datas passed to MAP_F
<i>A</i>	Generated on Tue Aug 21 2012 16:00:00 for gdsi by Doxygen

Returns

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

See also

gdsl_rbtrees_map_infix() (p. 203)
gdsl_rbtrees_map_postfix() (p. 204)

4.15.2.15 `gsdl_element_t gsdl_rbtrees_map_infix (const gsdl_rbtrees_t T,
 gsdl_map_func_t MAP_F, void * USER_DATA)`

Parse a red-black tree in infix order.

Parse all nodes of the red-black tree T in infix 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 **gsdl_rbtrees_map_infix()** (p. 203) stops and returns its last examined element.

Note

Complexity: $O(|T|)$

Precondition

T must be a valid gsdl_rbtrees_t & MAP_F != NULL

Parameters

<i>T</i>	The red-black tree to map.
<i>MAP_F</i>	The map function.
<i>USER_DATA</i>	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

gsdl_rbtrees_map_prefix() (p. 202)
gsdl_rbtrees_map_postfix() (p. 204)

4.15.2.16 `gdsi_element_t gdsi_rbtrees_map_postfix(const gdsi_rbtrees_t T, gdsi_map_func_t MAP_F, void * USER_DATA)`

Parse a red-black tree in postfix order.

Parse all nodes of the red-black tree T in postfix 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 **gdsi_rbtrees_map_postfix()** (p. 204) stops and returns its last examined element.

Note

Complexity: $O(|T|)$

Precondition

T must be a valid gdsi_rbtrees_t & MAP_F != NULL

Parameters

<i>T</i>	The red-black tree to map.
<i>MAP_F</i>	The map function.
<i>USER_DATA</i>	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

gdsi_rbtrees_map_prefix() (p. 202)
gdsi_rbtrees_map_infix() (p. 203)

4.15.2.17 `void gdsi_rbtrees_write(const gdsi_rbtrees_t T, gdsi_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_rbtrees_t` & `WRITE_F` != NULL & `OUTPUT_FILE` != NULL

Parameters

<i>T</i>	The red-black tree to write.
<i>WRITE_F</i>	The write function.
<i>OUTPUT_FILE</i>	The file where to write T's elements.
<i>USER_DATA</i>	User's datas passed to <code>WRITE_F</code> .

See also

`gdsl_rbtrees_write_xml()` (p. 205)

`gdsl_rbtrees_dump()` (p. 206)

4.15.2.18 `void gdsl_rbtrees_write_xml(const gdsl_rbtrees_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`.
Additional `USER_DATA` argument could be passed to `WRITE_F`.

Note

Complexity: $O(|T|)$

Precondition

T must be a valid `gdsl_rbtrees_t` & `OUTPUT_FILE` != NULL

Parameters

<i>T</i>	The red-black tree to write.
<i>WRITE_F</i>	The write function.
<i>OUTPUT_FILE</i>	The file where to write T's elements.
<i>USER_DATA</i>	User's datas passed to <code>WRITE_F</code> .

See also

`gdsl_rbtrees_write()` (p. 204)

`gdsl_rbtrees_dump()` (p. 206)

4.15.2.19 void **gdsi_rbtrees_dump**(const **gdsi_rbtrees_t** *T*, **gdsi_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 **gdsi_rbtrees_t** & *OUTPUT_FILE* != NULL

Parameters

<i>T</i>	The red-black tree to write.
<i>WRITE_F</i>	The write function.
<i>OUTPUT_FILE</i>	The file where to write <i>T</i> 's elements.
<i>USER_DATA</i>	User's datas passed to <i>WRITE_F</i> .

See also

gdsi_rbtrees_write() (p. 204)

gdsi_rbtrees_write_xml() (p. 205)

4.16 Sort module

Functions

- void **gdsi_sort** (**gdsi_element_t** **T*, **ulong** *N*, const **gdsi_compare_func_t** *COMP_F*)

Sort an array in place.

4.16.1 Function Documentation

4.16.1.1 void **gdsi_sort** (**gdsi_element_t** * *T*, **ulong** *N*, const **gdsi_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

$N == |T|$ & $T \neq \text{NULL}$ & $\text{COMP_F} \neq \text{NULL}$ & for all $i \leq N$: $\text{sizeof}(T[i]) == \text{sizeof}(\text{gdsi_element_t})$

Parameters

<i>T</i>	The array of elements to sort
<i>N</i>	The number of elements into <i>T</i>
<i>COMP_F</i>	The function pointer used to compare <i>T</i> 's elements

4.17 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)
Get the 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.

- **gdsI_element_t gdsI_stack_map_forward** (const **gdsI_stack_t** S, **gdsI_map_func_t** MAP_F, void *USER_DATA)

Parse a stack from bottom to top.

- **gdsI_element_t gdsI_stack_map_backward** (const **gdsI_stack_t** S, **gdsI_map_func_t** MAP_F, void *USER_DATA)

Parse a stack from top to bottom.

- void **gdsI_stack_write** (const **gdsI_stack_t** S, **gdsI_write_func_t** WRITE_F, FILE *OUTPUT_FILE, void *USER_DATA)

Write all the elements of a stack to a file.

- void **gdsI_stack_write_xml** (**gdsI_stack_t** S, **gdsI_write_func_t** WRITE_F, FILE *OUTPUT_FILE, void *USER_DATA)

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

- void **gdsI_stack_dump** (**gdsI_stack_t** S, **gdsI_write_func_t** WRITE_F, FILE *OUTPUT_FILE, void *USER_DATA)

Dump the internal structure of a stack to a file.

4.17.1 Typedef Documentation

4.17.1.1 typedef struct _gdsI_stack* gdsI_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 53 of file gdsI_stack.h.

4.17.2 Function Documentation

4.17.2.1 **gdsI_stack_t gdsI_stack_alloc** (const char * NAME, **gdsI_alloc_func_t** ALLOC_F, **gdsI_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

<i>NAME</i>	The name of the new stack to create
<i>ALLOC_F</i>	Function to alloc element when inserting it in a stack
<i>FREE_F</i>	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

gdsI_stack_free() (p. 210)
gdsI_stack_flush() (p. 211)

4.17.2.2 void gdsI_stack_free(gdsI_stack_t S)

Destroy a stack.

Deallocate all the elements of the stack S by calling S's FREE_F function passed to **gdsI_stack_alloc()** (p. 209). The name of S is deallocated and S is deallocated itself too.

Note

Complexity: $O(|S|)$

Precondition

S must be a valid gdsI_stack_t

Parameters

S	The stack to destroy
---	----------------------

See also

gdsI_stack_alloc() (p. 209)
gdsI_stack_flush() (p. 211)

4.17.2.3 void `gdsI_stack_flush(gdsI_stack_t S)`

Flush a stack.

Deallocate all the elements of the stack `S` by calling `S`'s `FREE_F` function passed to **`gdsI_stack_alloc()`** (p. 209). `S` is not deallocated itself and `S`'s name is not modified.

Note

Complexity: $O(|S|)$

Precondition

`S` must be a valid `gdsI_stack_t`

Parameters

<code>S</code>	The stack to flush
----------------	--------------------

See also

`gdsI_stack_alloc()` (p. 209)

`gdsI_stack_free()` (p. 210)

4.17.2.4 const char* `gdsI_stack_get_name(const gdsI_stack_t S)`

Get the name of a stack.

Note

Complexity: $O(1)$

Precondition

`Q` must be a valid `gdsI_stack_t`

Postcondition

The returned string **MUST NOT** be freed.

Parameters

<code>S</code>	The stack to get the name from
----------------	--------------------------------

Returns

the name of the stack S.

See also

gdsI_stack_set_name() (p. 214)

4.17.2.5 ulong gdsI_stack_get_size(const gdsI_stack_t S)

Get the size of a stack.

Note

Complexity: $O(1)$

Precondition

S must be a valid gdsI_stack_t

Parameters

S	The stack to get the size from
---	--------------------------------

Returns

the number of elements of the stack S (noted $|S|$).

4.17.2.6 ulong gdsI_stack_get_growing_factor(const gdsI_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 **gdsI_stack_set_growing_factor()** (p. 215).

Note

Complexity: $O(1)$

Precondition

S must be a valid gdsI_stack_t

Parameters

S	The stack to get the growing factor from
---	--

Returns

the growing factor of the stack S.

See also

gdsI_stack_insert() (p. 216)

gdsI_stack_set_growing_factor() (p. 215)

4.17.2.7 bool gdsI_stack_is_empty(const gdsI_stack_t S)

Check if a stack is empty.

Note

Complexity: $O(1)$

Precondition

S must be a valid gdsI_stack_t

Parameters

S	The stack to check
---	--------------------

Returns

TRUE if the stack S is empty.

FALSE if the stack S is not empty.

4.17.2.8 gdsI_element_t gdsI_stack_get_top(const gdsI_stack_t S)

Get the top of a stack.

Note

Complexity: $O(1)$

Precondition

S must be a valid gdsI_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

gdsI_stack_get_bottom() (p. 214)

4.17.2.9 gdsI_element_t gdsI_stack_get_bottom(const gdsI_stack_t S)

Get the bottom of a stack.

Note

Complexity: $O(1)$

Precondition

S must be a valid gdsI_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

gdsI_stack_get_top() (p. 213)

4.17.2.10 gdsI_stack_t gdsI_stack_set_name(gdsI_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. 211)

4.17.2.11 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. 212)

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

gdsI_stack_insert() (p. 216)

gdsI_stack_get_growing_factor() (p. 212)

4.17.2.12 gdsI_element_t gdsI_stack_insert(gdsI_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 **gdsI_stack_alloc()** (p. 209). 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 gdsI_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

gdsI_stack_set_growing_factor() (p. 215)

gdsI_stack_get_growing_factor() (p. 212)

gdsI_stack_remove() (p. 216)

4.17.2.13 gdsI_element_t gdsI_stack_remove(gdsI_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. 216)

4.17.2.14 `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.218)

4.17.2.15 `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` & $POS > 0$ & $POS \leq |S|$

Parameters

<i>S</i>	The stack to search the element in
<i>POS</i>	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 $POS \leq 0$.

See also

gdsl_stack_search() (p.217)

4.17.2.16 `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.218) stops and returns its last examined element.

Note

Complexity: $O(|S|)$

Precondition

S must be a valid `gdsl_stack_t` & `MAP_F` != NULL

Parameters

<code>S</code>	The stack to parse
<code>MAP_F</code>	The map function to apply on each S's element
<code>USER_DATA</code>	User's datas passed to <code>MAP_F</code> Returns the first element for which <code>MAP_F</code> returns <code>GDSL_MAP_STOP</code> . Returns NULL when the parsing is done.

See also

`gdsl_stack_map_backward()` (p. 219)

4.17.2.17 **`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. 219) stops and returns its last examined element.

Note

Complexity: $O(|S|)$

Precondition

S must be a valid `gdsl_stack_t` & `MAP_F` != NULL

Parameters

<code>S</code>	The stack to parse
<code>MAP_F</code>	The map function to apply on each S's element
<code>USER_DATA</code>	User's datas passed to <code>MAP_F</code>

Returns

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

See also

gdsI_stack_map_forward() (p. 218)

4.17.2.18 void **gdsI_stack_write**(const **gdsI_stack_t** *S*, **gdsI_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. -
Additional USER_DATA argument could be passed to *WRITE_F*.

Note

Complexity: $O(|S|)$

Precondition

S must be a valid **gdsI_stack_t** & *OUTPUT_FILE* != NULL & *WRITE_F* != NULL

Parameters

<i>S</i>	The stack to write.
<i>WRITE_F</i>	The write function.
<i>OUTPUT_FILE</i>	The file where to write <i>S</i> 's elements.
<i>USER_DATA</i>	User's datas passed to <i>WRITE_F</i> .

See also

gdsI_stack_write_xml() (p. 220)

gdsI_stack_dump() (p. 221)

4.17.2.19 void **gdsI_stack_write_xml**(**gdsI_stack_t** *S*, **gdsI_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*. Additional USER_DATA argument could be passed to *WRITE_F*.

Note

Complexity: $O(|S|)$

Precondition

S must be a valid `gdsI_stack_t` & `OUTPUT_FILE` != NULL

Parameters

<code>S</code>	The stack to write.
<code>WRITE_F</code>	The write function.
<code>OUTPUT_FILE</code>	The file where to write S's elements.
<code>USER_DATA</code>	User's datas passed to <code>WRITE_F</code> .

See also

`gdsI_stack_write()` (p. 220)

`gdsI_stack_dump()` (p. 221)

4.17.2.20 `void gdsI_stack_dump(gdsI_stack_t S, gdsI_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 `gdsI_stack_t` & `OUTPUT_FILE` != NULL

Parameters

<code>S</code>	The stack to write.
<code>WRITE_F</code>	The write function.
<code>OUTPUT_FILE</code>	The file where to write S's elements.
<code>USER_DATA</code>	User's datas passed to <code>WRITE_F</code> .

See also

`gdsI_stack_write()` (p. 220)

`gdsI_stack_write_xml()` (p. 220)

4.18 GDSDL types

Typedefs

- typedef void * **gdsl_element_t**
GDSDL element type.
- typedef **gdsl_element_t**(* **gdsl_alloc_func_t**)(void *USER_DATA)
GDSDL Alloc element function type.
- typedef void(* **gdsl_free_func_t**)(**gdsl_element_t** E)
GDSDL Free element function type.
- typedef **gdsl_element_t**(* **gdsl_copy_func_t**)(const **gdsl_element_t** E)
GDSDL Copy element function type.
- typedef int(* **gdsl_map_func_t**)(const **gdsl_element_t** E, **gdsl_location_t** LOCATION, void *USER_DATA)
GDSDL Map element function type.
- typedef long int(* **gdsl_compare_func_t**)(const **gdsl_element_t** E, void *VALUE)
GDSDL 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)
GDSDL Write element function type.
- typedef unsigned long int **ulong**
- typedef unsigned short int **ushort**

Enumerations

- enum **gdsl_constant_t** { **GDSDL_ERR_MEM_ALLOC** = -1, **GDSDL_MAP_STOP** = 0, **GDSDL_MAP_CONT** = 1, **GDSDL_INSERTED**, **GDSDL_FOUND** }
GDSDL Constants.
- enum **gdsl_location_t** { **GDSDL_LOCATION_UNDEF** = 0, **GDSDL_LOCATION_HEAD** = 1, **GDSDL_LOCATION_ROOT** = 1, **GDSDL_LOCATION_TOP** = 1, **GDSDL_LOCATION_TAIL** = 2, **GDSDL_LOCATION_LEAF** = 2, **GDSDL_LOCATION_BOTTOM** = 2, **GDSDL_LOCATION_FIRST** = 1, **GDSDL_LOCATION_LAST** = 2, **GDSDL_LOCATION_FIRST_COL** = 1, **GDSDL_LOCATION_LAST_COL** = 2, **GDSDL_LOCATION_FIRST_ROW** = 4, **GDSDL_LOCATION_LAST_ROW** = 8 }
- enum **bool** { **FALSE** = 0, **TRUE** = 1 }

4.18.1 Typedef Documentation

4.18.1.1 typedef void* gdsl_element_t

GDSDL element type.

All GDSDL internal data structures contains a field of this type. This field is for GDSDL users to store their data into GDSDL data structures.

Definition at line 130 of file `gdsl_types.h`.

4.18.1.2 `typedef gdsl_element_t(*gdsl_alloc_func_t)(void *USER_DATA)`

GDSDL 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

<code>USER_DATA</code>	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. 223)

Definition at line 144 of file `gdsl_types.h`.

4.18.1.3 `typedef void(*gdsl_free_func_t)(gdsl_element_t E)`

GDSDL 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 resources allocated by the corresponding call to the function of type `gdsl_alloc_func_t`. The GDSDL functions doesn't check if `E != NULL` before calling this function.

Parameters

<code>E</code>	The element to deallocate.
----------------	----------------------------

See also

`gdsl_alloc_func_t` (p. 223)

Definition at line 162 of file `gdsl_types.h`.

4.18.1.4 `typedef gdsl_element_t(*gdsl_copy_func_t)(const gdsl_element_t E)`

GDSDL Copy element function type.

This function type is for copying `gdsl_element_t` variables.

Parameters

<i>E</i>	The <code>gdsi_element_t</code> variable to copy.
----------	---

Returns

the copied element in case of success.
 NULL in case of failure.

Definition at line 175 of file `gdsi_types.h`.

4.18.1.5 `typedef int(* gdsi_map_func_t)(const gdsi_element_t E, gdsi_location_t LOCATION, void *USER_DATA)`

GDSL Map element function type.

This function type is for mapping a `gdsi_element_t` variable from a GDSL data structure. The optional `USER_DATA` could be used to do special thing if needed.

Parameters

<i>E</i>	The actually mapped <code>gdsi_element_t</code> variable.
<i>LOCATION</i>	The location of <i>E</i> in the data structure.
<i>USER_DATA</i>	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 192 of file `gdsi_types.h`.

4.18.1.6 `typedef long int(* gdsi_compare_func_t)(const gdsi_element_t E, void *VALUE)`

GDSL Comparison element function type.

This function type is used to compare a `gdsi_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

<i>E</i>	The <code>gdsi_element_t</code> variable contained into the data structure to compare from.
<i>VALUE</i>	The user data to compare <i>E</i> 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 213 of file gdsI_types.h.

4.18.1.7 `typedef void(* gdsI_write_func_t)(const gdsI_element_t E, FILE *OUTPUT_FILE, gdsI_location_t LOCATION, void *USER_DATA)`

GDSDL Write element function type.

This function type is for writing a `gdsI_element_t` E to `OUTPUT_FILE`. Additional `USER_DATA` could be passed to it.

Parameters

<i>E</i>	The gdsI element to write.
<i>OUTPUT_FILE</i>	The file where to write E.
<i>LOCATION</i>	The location of E in the data structure.
<i>USER_DATA</i>	User's datas.

Definition at line 229 of file gdsI_types.h.

4.18.1.8 `typedef unsigned long int ulong`

Definition at line 242 of file gdsI_types.h.

4.18.1.9 `typedef unsigned short int ushort`

Definition at line 246 of file gdsI_types.h.

4.18.2 Enumeration Type Documentation

4.18.2.1 `enum gdsI_constant_t`

GDSDL Constants.

Enumerator:

- GDSDL_ERR_MEM_ALLOC*** Memory allocation error
- GDSDL_MAP_STOP*** For stopping a parsing function
- GDSDL_MAP_CONT*** For continuing a parsing function
- GDSDL_INSERTED*** To indicate an inserted value

GDSL_FOUND To indicate a founded value

Definition at line 48 of file gdsl_types.h.

4.18.2.2 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 69 of file gdsl_types.h.

4.18.2.3 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 267 of file gdsl_types.h.

Chapter 5

File Documentation

5.1 `_gdsl_bintree.h` File Reference

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 infix 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 postfix 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.

5.2 `_gdsl_bstree.h` File Reference

Typedefs

- typedef `_gdsl_bintree_t` `_gdsl_bstree_t`
GDSL low-level binary search tree type.
- typedef int(* `_gdsl_bstree_map_func_t`)(_gdsl_bstree_t TREE, void *USER_DATA)
GDSL low-level binary search tree map function type.
- 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.

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 _gsdl_bstree_get_right** (const _gsdl_bstree_t T)

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

- **ulong _gsdl_bstree_get_size** (const _gsdl_bstree_t T)

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

- **ulong _gsdl_bstree_get_height** (const _gsdl_bstree_t T)

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

- **_gsdl_bstree_t _gsdl_bstree_insert** (_gsdl_bstree_t *T, const gsdl_compare_func_t COMP_F, const gsdl_element_t VALUE, int *RESULT)

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

- **gsdl_element_t _gsdl_bstree_remove** (_gsdl_bstree_t *T, const gsdl_compare_func_t COMP_F, const gsdl_element_t VALUE)

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

- **_gsdl_bstree_t _gsdl_bstree_search** (const _gsdl_bstree_t T, const gsdl_compare_func_t COMP_F, const gsdl_element_t VALUE)

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

- **_gsdl_bstree_t _gsdl_bstree_search_next** (const _gsdl_bstree_t T, const gsdl_compare_func_t COMP_F, const gsdl_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.

- **_gsdl_bstree_t _gsdl_bstree_map_prefix** (const _gsdl_bstree_t T, const _gsdl_bstree_map_func_t MAP_F, void *USER_DATA)

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

- **_gsdl_bstree_t _gsdl_bstree_map_infix** (const _gsdl_bstree_t T, const _gsdl_bstree_map_func_t MAP_F, void *USER_DATA)

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

- **_gsdl_bstree_t _gsdl_bstree_map_postfix** (const _gsdl_bstree_t T, const _gsdl_bstree_map_func_t MAP_F, void *USER_DATA)

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

- **void _gsdl_bstree_write** (const _gsdl_bstree_t T, const _gsdl_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 _gsdl_bstree_write_xml** (const _gsdl_bstree_t T, const _gsdl_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 _gsdl_bstree_dump** (const _gsdl_bstree_t T, const _gsdl_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.

5.3 _gsdl_list.h File Reference

Typedefs

- **typedef _gsdl_node_t _gsdl_list_t**
GDSL low-level doubly-linked list type.

Functions

- **_gdsi_list_t _gdsi_list_alloc** (const **gdsi_element_t** E)
Create a new low-level list.
- **void _gdsi_list_free** (_gdsi_list_t L, const **gdsi_free_func_t** FREE_F)
Destroy a low-level list.
- **bool _gdsi_list_is_empty** (const _gdsi_list_t L)
Check if a low-level list is empty.
- **ulong _gdsi_list_get_size** (const _gdsi_list_t L)
Get the size of a low-level list.
- **void _gdsi_list_link** (_gdsi_list_t L1, _gdsi_list_t L2)
Link two low-level lists together.
- **void _gdsi_list_insert_after** (_gdsi_list_t L, _gdsi_list_t PREV)
Insert a low-level list after another one.
- **void _gdsi_list_insert_before** (_gdsi_list_t L, _gdsi_list_t SUCC)
Insert a low-level list before another one.
- **void _gdsi_list_remove** (_gdsi_node_t NODE)
Remove a node from a low-level list.
- **_gdsi_list_t _gdsi_list_search** (_gdsi_list_t L, const **gdsi_compare_func_t** COMP_F, void *VALUE)
Search for a particular node in a low-level list.
- **_gdsi_list_t _gdsi_list_map_forward** (const _gdsi_list_t L, const **gdsi_node_map_func_t** MAP_F, void *USER_DATA)
Parse a low-level list in forward order.
- **_gdsi_list_t _gdsi_list_map_backward** (const _gdsi_list_t L, const **gdsi_node_map_func_t** MAP_F, void *USER_DATA)
Parse a low-level list in backward order.
- **void _gdsi_list_write** (const _gdsi_list_t L, const **gdsi_node_write_func_t** WRITE_F, FILE *OUTPUT_FILE, void *USER_DATA)
Write all nodes of a low-level list to a file.
- **void _gdsi_list_write_xml** (const _gdsi_list_t L, const **gdsi_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 _gdsi_list_dump** (const _gdsi_list_t L, const **gdsi_node_write_func_t** WRITE_F, FILE *OUTPUT_FILE, void *USER_DATA)
Dump the internal structure of a low-level list to a file.

5.4 _gdsi_node.h File Reference

Typedefs

- **typedef struct _gdsi_node * _gdsi_node_t**
GDSL low-level doubly linked node type.

- `typedef int(* _gdsi_node_map_func_t)(const _gdsi_node_t NODE, void *USER_DATA)`
GDSL low-level doubly-linked node map function type.
- `typedef void(* _gdsi_node_write_func_t)(const _gdsi_node_t NODE, FILE *OUTPUT_FILE, void *USER_DATA)`
GDSL low-level doubly-linked node write function type.

Functions

- `_gdsi_node_t _gdsi_node_alloc (void)`
Create a new low-level node.
- `gdsi_element_t _gdsi_node_free (_gdsi_node_t NODE)`
Destroy a low-level node.
- `_gdsi_node_t _gdsi_node_get_succ (const _gdsi_node_t NODE)`
Get the successor of a low-level node.
- `_gdsi_node_t _gdsi_node_get_pred (const _gdsi_node_t NODE)`
Get the predecessor of a low-level node.
- `gdsi_element_t _gdsi_node_get_content (const _gdsi_node_t NODE)`
Get the content of a low-level node.
- `void _gdsi_node_set_succ (_gdsi_node_t NODE, const _gdsi_node_t SUC-C)`
Set the successor of a low-level node.
- `void _gdsi_node_set_pred (_gdsi_node_t NODE, const _gdsi_node_t PRE-D)`
Set the predecessor of a low-level node.
- `void _gdsi_node_set_content (_gdsi_node_t NODE, const gdsi_element_t -CONTENT)`
Set the content of a low-level node.
- `void _gdsi_node_link (_gdsi_node_t NODE1, _gdsi_node_t NODE2)`
Link two low-level nodes together.
- `void _gdsi_node_unlink (_gdsi_node_t NODE1, _gdsi_node_t NODE2)`
Unlink two low-level nodes.
- `void _gdsi_node_write (const _gdsi_node_t NODE, const _gdsi_node_write_func_t WRITE_F, FILE *OUTPUT_FILE, void *USER_DATA)`
Write a low-level node to a file.
- `void _gdsi_node_write_xml (const _gdsi_node_t NODE, const _gdsi_node_write_func_t WRITE_F, FILE *OUTPUT_FILE, void *USER_DATA)`
Write a low-level node to a file into XML.
- `void _gdsi_node_dump (const _gdsi_node_t NODE, const _gdsi_node_write_func_t WRITE_F, FILE *OUTPUT_FILE, void *USER_DATA)`
Dump the internal structure of a low-level node to a file.

5.5 gdsl.h File Reference

Functions

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

5.6 gdsl_2darray.h File Reference

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 *NEW_NAME)
Set the name of a 2D-array.
- **gdsl_element_t gdsl_2darray_set_content** (gdsl_2darray_t A, const **ulong** R, const **ulong** C, void *VALUE)
Modify an element in a 2D-array.
- **void gdsl_2darray_write** (const gdsl_2darray_t A, const **gdsl_write_func_t** WRITE_F, FILE *OUTPUT_FILE, void *USER_DATA)
Write the content of a 2D-array to a file.

- void **gdsl_2darray_write_xml** (const **gdsl_2darray_t** A, const **gdsl_write_func_t** WRITE_F, FILE *OUTPUT_FILE, void *USER_DATA)
Write the content of a 2D array to a file into XML.
- void **gdsl_2darray_dump** (const **gdsl_2darray_t** A, const **gdsl_write_func_t** WRITE_F, FILE *OUTPUT_FILE, void *USER_DATA)
Dump the internal structure of a 2D array to a file.

5.7 gdsl_bstree.h File Reference

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_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 infix 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 postfix 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.

5.8 gdsl_hash.h File Reference

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 **gdsI_hash_flush** (gdsI_hash_t H)
Flush a hashtable.
- const char * **gdsI_hash_get_name** (const gdsI_hash_t H)
Get the name of a hashtable.
- ushort **gdsI_hash_get_entries_number** (const gdsI_hash_t H)
Get the number of entries of a hashtable.
- ushort **gdsI_hash_get_lists_max_size** (const gdsI_hash_t H)
Get the max number of elements allowed in each entry of a hashtable.
- ushort **gdsI_hash_get_longest_list_size** (const gdsI_hash_t H)
Get the number of elements of the longest list entry of a hashtable.
- ulong **gdsI_hash_get_size** (const gdsI_hash_t H)
Get the size of a hashtable.
- double **gdsI_hash_get_fill_factor** (const gdsI_hash_t H)
Get the fill factor of a hashtable.
- gdsI_hash_t **gdsI_hash_set_name** (gdsI_hash_t H, const char *NEW_NAME)
Set the name of a hashtable.
- gdsI_element_t **gdsI_hash_insert** (gdsI_hash_t H, void *VALUE)
Insert an element into a hashtable (PUSH).
- gdsI_element_t **gdsI_hash_remove** (gdsI_hash_t H, const char *KEY)
Remove an element from a hashtable (POP).
- gdsI_hash_t **gdsI_hash_delete** (gdsI_hash_t H, const char *KEY)
Delete an element from a hashtable.
- gdsI_hash_t **gdsI_hash_modify** (gdsI_hash_t H, ushort NEW_ENTRIES_NB, ushort NEW_LISTS_MAX_SIZE)
Increase the dimensions of a hashtable.
- gdsI_element_t **gdsI_hash_search** (const gdsI_hash_t H, const char *KEY)
Search for a particular element into a hashtable (GET).
- gdsI_element_t **gdsI_hash_map** (const gdsI_hash_t H, gdsI_map_func_t MAP_F, void *USER_DATA)
Parse a hashtable.
- void **gdsI_hash_write** (const gdsI_hash_t H, gdsI_write_func_t WRITE_F, FILE *OUTPUT_FILE, void *USER_DATA)
Write all the elements of a hashtable to a file.
- void **gdsI_hash_write_xml** (const gdsI_hash_t H, gdsI_write_func_t WRITE_F, FILE *OUTPUT_FILE, void *USER_DATA)
Write the content of a hashtable to a file into XML.
- void **gdsI_hash_dump** (const gdsI_hash_t H, gdsI_write_func_t WRITE_F, FILE *OUTPUT_FILE, void *USER_DATA)
Dump the internal structure of a hashtable to a file.

5.9 gdsl_heap.h File Reference

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_NAME)
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, FILE *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 **gdsi_heap_dump** (const **gdsi_heap_t** H, **gdsi_write_func_t** WRITE_F, FILE *OUTPUT_FILE, void *USER_DATA)

Dump the internal structure of a heap to a file.

5.10 gdsi_interval_heap.h File Reference

Typedefs

- typedef struct heap * **gdsi_interval_heap_t**

GDSL interval heap type.

Functions

- **gdsi_interval_heap_t gdsi_interval_heap_alloc** (const char *NAME, **gdsi_alloc_func_t** ALLOC_F, **gdsi_free_func_t** FREE_F, **gdsi_compare_func_t** COMP_F)

Create a new interval heap.

- void **gdsi_interval_heap_free** (**gdsi_interval_heap_t** H)

Destroy an interval heap.

- void **gdsi_interval_heap_flush** (**gdsi_interval_heap_t** H)

Flush an interval heap.

- const char * **gdsi_interval_heap_get_name** (const **gdsi_interval_heap_t** H)

Get the name of an interval heap.

- **ulong gdsi_interval_heap_get_size** (const **gdsi_interval_heap_t** H)

Get the size of a interval heap.

- void **gdsi_interval_heap_set_max_size** (const **gdsi_interval_heap_t** H, **ulong** size)

Set the maximum size of the interval heap.

- **bool gdsi_interval_heap_is_empty** (const **gdsi_interval_heap_t** H)

Check if an interval heap is empty.

- **gdsi_interval_heap_t gdsi_interval_heap_set_name** (**gdsi_interval_heap_t** H, const char *NEW_NAME)

Set the name of an interval heap.

- **gdsi_element_t gdsi_interval_heap_insert** (**gdsi_interval_heap_t** H, void *VALUE)

Insert an element into an interval heap (PUSH).

- **gdsi_element_t gdsi_interval_heap_remove_max** (**gdsi_interval_heap_t** H)

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

- **gdsi_element_t gdsi_interval_heap_remove_min** (**gdsi_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.

5.11 gdsl_list.h File Reference

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 * gdsi_list_get_name** (const **gdsi_list_t** L)
Get the name of a list.
- **ulong gdsi_list_get_size** (const **gdsi_list_t** L)
Get the size of a list.
- **bool gdsi_list_is_empty** (const **gdsi_list_t** L)
Check if a list is empty.
- **gdsi_element_t gdsi_list_get_head** (const **gdsi_list_t** L)
Get the head of a list.
- **gdsi_element_t gdsi_list_get_tail** (const **gdsi_list_t** L)
Get the tail of a list.
- **gdsi_list_t gdsi_list_set_name** (**gdsi_list_t** L, const char *NEW_NAME)
Set the name of a list.
- **gdsi_element_t gdsi_list_insert_head** (**gdsi_list_t** L, void *VALUE)
Insert an element at the head of a list.
- **gdsi_element_t gdsi_list_insert_tail** (**gdsi_list_t** L, void *VALUE)
Insert an element at the tail of a list.
- **gdsi_element_t gdsi_list_remove_head** (**gdsi_list_t** L)
Remove the head of a list.
- **gdsi_element_t gdsi_list_remove_tail** (**gdsi_list_t** L)
Remove the tail of a list.
- **gdsi_element_t gdsi_list_remove** (**gdsi_list_t** L, **gdsi_compare_func_t** COMP_F, const void *VALUE)
Remove a particular element from a list.
- **gdsi_list_t gdsi_list_delete_head** (**gdsi_list_t** L)
Delete the head of a list.
- **gdsi_list_t gdsi_list_delete_tail** (**gdsi_list_t** L)
Delete the tail of a list.
- **gdsi_list_t gdsi_list_delete** (**gdsi_list_t** L, **gdsi_compare_func_t** COMP_F, const void *VALUE)
Delete a particular element from a list.
- **gdsi_element_t gdsi_list_search** (const **gdsi_list_t** L, **gdsi_compare_func_t** COMP_F, const void *VALUE)
Search for a particular element into a list.
- **gdsi_element_t gdsi_list_search_by_position** (const **gdsi_list_t** L, ulong POS)
Search for an element by its position in a list.
- **gdsi_element_t gdsi_list_search_max** (const **gdsi_list_t** L, **gdsi_compare_func_t** COMP_F)
Search for the greatest element of a list.
- **gdsi_element_t gdsi_list_search_min** (const **gdsi_list_t** L, **gdsi_compare_func_t** COMP_F)
Search for the lowest element of a list.
- **gdsi_list_t gdsi_list_sort** (**gdsi_list_t** L, **gdsi_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, FILE *OUTPUT_FILE, void *USER_DATA)
Write the content of a list to a file into XML.
- void **gdsl_list_dump** (const **gdsl_list_t** L, **gdsl_write_func_t** WRITE_F, FILE *OUTPUT_FILE, void *USER_DATA)
Dump the internal structure of a list to a file.
- **gdsl_list_cursor_t gdsl_list_cursor_alloc** (const **gdsl_list_t** L)
Create a new list cursor.
- void **gdsl_list_cursor_free** (**gdsl_list_cursor_t** C)
Destroy a list cursor.
- void **gdsl_list_cursor_move_to_head** (**gdsl_list_cursor_t** C)
Put a cursor on the head of its list.
- void **gdsl_list_cursor_move_to_tail** (**gdsl_list_cursor_t** C)
Put a cursor on the tail of its list.
- **gdsl_element_t gdsl_list_cursor_move_to_value** (**gdsl_list_cursor_t** C, **gdsl_compare_func_t** COMP_F, void *VALUE)
Place a cursor on a particular element.
- **gdsl_element_t gdsl_list_cursor_move_to_position** (**gdsl_list_cursor_t** C, **ulong** POS)
Place a cursor on a element given by its position.
- void **gdsl_list_cursor_step_forward** (**gdsl_list_cursor_t** C)
Move a cursor one step forward of its list.
- void **gdsl_list_cursor_step_backward** (**gdsl_list_cursor_t** C)
Move a cursor one step backward of its list.
- **bool gdsl_list_cursor_is_on_head** (const **gdsl_list_cursor_t** C)
Check if a cursor is on the head of its list.
- **bool gdsl_list_cursor_is_on_tail** (const **gdsl_list_cursor_t** C)
Check if a cursor is on the tail of its list.
- **bool gdsl_list_cursor_has_succ** (const **gdsl_list_cursor_t** C)
Check if a cursor has a successor.
- **bool gdsl_list_cursor_has_pred** (const **gdsl_list_cursor_t** C)
Check if a cursor has a predecessor.
- void **gdsl_list_cursor_set_content** (**gdsl_list_cursor_t** C, **gdsl_element_t** E)
Set the content of the cursor.

- **gdsl_element_t gsdl_list_cursor_get_content** (const **gsdl_list_cursor_t** C)
Get the content of a cursor.
- **gsdl_element_t gsdl_list_cursor_insert_after** (**gsdl_list_cursor_t** C, void *VALUE)
Insert a new element after a cursor.
- **gsdl_element_t gsdl_list_cursor_insert_before** (**gsdl_list_cursor_t** C, void *VALUE)
Insert a new element before a cursor.
- **gsdl_element_t gsdl_list_cursor_remove** (**gsdl_list_cursor_t** C)
Remove the element under a cursor.
- **gsdl_element_t gsdl_list_cursor_remove_after** (**gsdl_list_cursor_t** C)
Remove the element after a cursor.
- **gsdl_element_t gsdl_list_cursor_remove_before** (**gsdl_list_cursor_t** C)
Remove the element before a cursor.
- **gsdl_list_cursor_t gsdl_list_cursor_delete** (**gsdl_list_cursor_t** C)
Delete the element under a cursor.
- **gsdl_list_cursor_t gsdl_list_cursor_delete_after** (**gsdl_list_cursor_t** C)
Delete the element after a cursor.
- **gsdl_list_cursor_t gsdl_list_cursor_delete_before** (**gsdl_list_cursor_t** C)
Delete the element before the cursor of a list.

5.12 gsdl_macros.h File Reference

Defines

- #define **GDSL_MAX**(X, Y) (X>Y?X:Y)
Give the greatest number of two numbers.
- #define **GDSL_MIN**(X, Y) (X>Y?Y:X)
Give the lowest number of two numbers.

5.13 gsdl_perm.h File Reference

Typedefs

- typedef struct gsdl_perm * **gsdl_perm_t**
GDSL permutation type.
- typedef void(* **gsdl_perm_write_func_t**)(ulong E, FILE *OUTPUT_FILE, **gsdl_location_t** POSITION, void *USER_DATA)
GDSL permutation write function type.
- typedef struct gsdl_perm_data * **gsdl_perm_data_t**

Enumerations

- enum **gdsI_perm_position_t** { **GDSL_PERM_POSITION_FIRST** = 1, **GDSL_PERM_POSITION_LAST** = 2 }

This type is for gdsI_perm_write_func_t.

Functions

- **gdsI_perm_t gdsI_perm_alloc** (const char *NAME, const **ulong** N)
Create a new permutation.
- void **gdsI_perm_free** (gdsI_perm_t P)
Destroy a permutation.
- **gdsI_perm_t gdsI_perm_copy** (const gdsI_perm_t P)
Copy a permutation.
- const char * **gdsI_perm_get_name** (const gdsI_perm_t P)
Get the name of a permutation.
- **ulong gdsI_perm_get_size** (const gdsI_perm_t P)
Get the size of a permutation.
- **ulong gdsI_perm_get_element** (const gdsI_perm_t P, const **ulong** INDIX)
Get the (INDIX+1)-th element from a permutation.
- **ulong * gdsI_perm_get_elements_array** (const gdsI_perm_t P)
Get the array elements of a permutation.
- **ulong gdsI_perm_linear_inversions_count** (const gdsI_perm_t P)
Count the inversions number into a linear permutation.
- **ulong gdsI_perm_linear_cycles_count** (const gdsI_perm_t P)
Count the cycles number into a linear permutation.
- **ulong gdsI_perm_canonical_cycles_count** (const gdsI_perm_t P)
Count the cycles number into a canonical permutation.
- **gdsI_perm_t gdsI_perm_set_name** (gdsI_perm_t P, const char *NEW_NAME)
Set the name of a permutation.
- **gdsI_perm_t gdsI_perm_linear_next** (gdsI_perm_t P)
Get the next permutation from a linear permutation.
- **gdsI_perm_t gdsI_perm_linear_prev** (gdsI_perm_t P)
Get the previous permutation from a linear permutation.
- **gdsI_perm_t gdsI_perm_set_elements_array** (gdsI_perm_t P, const **ulong** *ARRAY)
Initialize a permutation with an array of values.
- **gdsI_perm_t gdsI_perm_multiply** (gdsI_perm_t RESULT, const gdsI_perm_t ALPHA, const gdsI_perm_t BETA)
Multiply two permutations.
- **gdsI_perm_t gdsI_perm_linear_to_canonical** (gdsI_perm_t Q, const gdsI_perm_t P)
Convert a linear permutation to its canonical form.

- **gdsl_perm_t gsdl_perm_canonical_to_linear** (gsdl_perm_t Q, const gsdl_perm_t P)
Convert a canonical permutation to its linear form.
- **gsdl_perm_t gsdl_perm_inverse** (gsdl_perm_t P)
Inverse in place a permutation.
- **gsdl_perm_t gsdl_perm_reverse** (gsdl_perm_t P)
Reverse in place a permutation.
- **gsdl_perm_t gsdl_perm_randomize** (gsdl_perm_t P)
Randomize a permutation.
- **gsdl_element_t * gsdl_perm_apply_on_array** (gsdl_element_t *V, const gsdl_perm_t P)
Apply a permutation on to a vector.
- void **gsdl_perm_write** (const gsdl_perm_t P, const gsdl_write_func_t WRITE_F, FILE *OUTPUT_FILE, void *USER_DATA)
Write the elements of a permutation to a file.
- void **gsdl_perm_write_xml** (const gsdl_perm_t P, const gsdl_write_func_t WRITE_F, FILE *OUTPUT_FILE, void *USER_DATA)
Write the elements of a permutation to a file into XML.
- void **gsdl_perm_dump** (const gsdl_perm_t P, const gsdl_write_func_t WRITE_F, FILE *OUTPUT_FILE, void *USER_DATA)
Dump the internal structure of a permutation to a file.

5.14 gsdl_queue.h File Reference

Typedefs

- typedef struct _gsdl_queue * **gsdl_queue_t**
GDSL queue type.

Functions

- **gsdl_queue_t gsdl_queue_alloc** (const char *NAME, gsdl_alloc_func_t ALL_OC_F, gsdl_free_func_t FREE_F)
Create a new queue.
- void **gsdl_queue_free** (gsdl_queue_t Q)
Destroy a queue.
- void **gsdl_queue_flush** (gsdl_queue_t Q)
Flush a queue.
- const char * **gsdl_queue_get_name** (const gsdl_queue_t Q)
Get the name of a queue.
- **ulong gsdl_queue_get_size** (const gsdl_queue_t Q)
Get the size of a queue.
- **bool gsdl_queue_is_empty** (const gsdl_queue_t Q)

Check if a queue is empty.

- **gdsI_element_t gdsI_queue_get_head** (const **gdsI_queue_t** Q)

Get the head of a queue.

- **gdsI_element_t gdsI_queue_get_tail** (const **gdsI_queue_t** Q)

Get the tail of a queue.

- **gdsI_queue_t gdsI_queue_set_name** (**gdsI_queue_t** Q, const char *NEW_NAME)

Set the name of a queue.

- **gdsI_element_t gdsI_queue_insert** (**gdsI_queue_t** Q, void *VALUE)

Insert an element in a queue (PUT).

- **gdsI_element_t gdsI_queue_remove** (**gdsI_queue_t** Q)

Remove an element from a queue (GET).

- **gdsI_element_t gdsI_queue_search** (const **gdsI_queue_t** Q, **gdsI_compare_func_t** COMP_F, void *VALUE)

Search for a particular element in a queue.

- **gdsI_element_t gdsI_queue_search_by_position** (const **gdsI_queue_t** Q, **ulong** POS)

Search for an element by its position in a queue.

- **gdsI_element_t gdsI_queue_map_forward** (const **gdsI_queue_t** Q, **gdsI_map_func_t** MAP_F, void *USER_DATA)

Parse a queue from head to tail.

- **gdsI_element_t gdsI_queue_map_backward** (const **gdsI_queue_t** Q, **gdsI_map_func_t** MAP_F, void *USER_DATA)

Parse a queue from tail to head.

- void **gdsI_queue_write** (const **gdsI_queue_t** Q, **gdsI_write_func_t** WRITE_F, FILE *OUTPUT_FILE, void *USER_DATA)

Write all the elements of a queue to a file.

- void **gdsI_queue_write_xml** (const **gdsI_queue_t** Q, **gdsI_write_func_t** WRITE_F, FILE *OUTPUT_FILE, void *USER_DATA)

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

- void **gdsI_queue_dump** (const **gdsI_queue_t** Q, **gdsI_write_func_t** WRITE_F, FILE *OUTPUT_FILE, void *USER_DATA)

Dump the internal structure of a queue to a file.

5.15 gdsI_rbtrees.h File Reference

Typedefs

- typedef struct gdsI_rbtrees * **gdsI_rbtrees_t**

Functions

- **gdsl_rbtrees_t gsdl_rbtrees_alloc** (const char *NAME, gsdl_alloc_func_t ALL-OC_F, gsdl_free_func_t FREE_F, gsdl_compare_func_t COMP_F)
Create a new red-black tree.
- void **gsdl_rbtrees_free** (gsdl_rbtrees_t T)
Destroy a red-black tree.
- void **gsdl_rbtrees_flush** (gsdl_rbtrees_t T)
Flush a red-black tree.
- char * **gsdl_rbtrees_get_name** (const gsdl_rbtrees_t T)
Get the name of a red-black tree.
- bool **gsdl_rbtrees_is_empty** (const gsdl_rbtrees_t T)
Check if a red-black tree is empty.
- gsdl_element_t **gsdl_rbtrees_get_root** (const gsdl_rbtrees_t T)
Get the root of a red-black tree.
- ulong **gsdl_rbtrees_get_size** (const gsdl_rbtrees_t T)
Get the size of a red-black tree.
- ulong **gsdl_rbtrees_height** (const gsdl_rbtrees_t T)
Get the height of a red-black tree.
- gsdl_rbtrees_t **gsdl_rbtrees_set_name** (gsdl_rbtrees_t T, const char *NEW_NAME)
Set the name of a red-black tree.
- gsdl_element_t **gsdl_rbtrees_insert** (gsdl_rbtrees_t T, void *VALUE, int *RESULT)
Insert an element into a red-black tree if it's not found or return it.
- gsdl_element_t **gsdl_rbtrees_remove** (gsdl_rbtrees_t T, void *VALUE)
Remove an element from a red-black tree.
- gsdl_rbtrees_t **gsdl_rbtrees_delete** (gsdl_rbtrees_t T, void *VALUE)
Delete an element from a red-black tree.
- gsdl_element_t **gsdl_rbtrees_search** (const gsdl_rbtrees_t T, gsdl_compare_func_t COMP_F, void *VALUE)
Search for a particular element into a red-black tree.
- gsdl_element_t **gsdl_rbtrees_map_prefix** (const gsdl_rbtrees_t T, gsdl_map_func_t MAP_F, void *USER_DATA)
Parse a red-black tree in prefixed order.
- gsdl_element_t **gsdl_rbtrees_map_infix** (const gsdl_rbtrees_t T, gsdl_map_func_t MAP_F, void *USER_DATA)
Parse a red-black tree in infix order.
- gsdl_element_t **gsdl_rbtrees_map_postfix** (const gsdl_rbtrees_t T, gsdl_map_func_t MAP_F, void *USER_DATA)
Parse a red-black tree in postfix order.
- void **gsdl_rbtrees_write** (const gsdl_rbtrees_t T, gsdl_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_rbtrees_write_xml** (const **gdsl_rbtrees_t** T, **gdsl_write_func_t** WRITE_F, FILE *OUTPUT_FILE, void *USER_DATA)
Write the content of a red-black tree to a file into XML.
- void **gdsl_rbtrees_dump** (const **gdsl_rbtrees_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.

5.16 gdsl_sort.h File Reference

Functions

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

5.17 gdsl_stack.h File Reference

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)
Get the 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, FILE *OUTPUT_FILE, void *USER_DATA)

Write all the elements of a stack to a file.

- void **gdsl_stack_write_xml** (**gdsl_stack_t** S, **gdsl_write_func_t** WRITE_F, FILE *OUTPUT_FILE, void *USER_DATA)

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

- void **gdsl_stack_dump** (**gdsl_stack_t** S, **gdsl_write_func_t** WRITE_F, FILE *OUTPUT_FILE, void *USER_DATA)

Dump the internal structure of a stack to a file.

5.18 gdsl_types.h File Reference

Typedefs

- typedef void * **gdsl_element_t**
GDSL element type.
- typedef **gdsl_element_t**(* **gdsl_alloc_func_t**)(void *USER_DATA)
GDSL Alloc element function type.
- typedef void(* **gdsl_free_func_t**)(**gdsl_element_t** E)
GDSL Free element function type.
- typedef **gdsl_element_t**(* **gdsl_copy_func_t**)(const **gdsl_element_t** E)

GDSL Copy element function type.

- typedef int(* **gdsl_map_func_t**)(const **gdsl_element_t** E, **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 }

5.19 mainpage.h File Reference

Index

Binary search tree manipulation module,
73

- gdsl_bstree_alloc, 74
- gdsl_bstree_delete, 81
- gdsl_bstree_dump, 86
- gdsl_bstree_flush, 75
- gdsl_bstree_free, 75
- gdsl_bstree_get_height, 78
- gdsl_bstree_get_name, 76
- gdsl_bstree_get_root, 77
- gdsl_bstree_get_size, 78
- gdsl_bstree_insert, 79
- gdsl_bstree_is_empty, 77
- gdsl_bstree_map_infix, 83
- gdsl_bstree_map_postfix, 84
- gdsl_bstree_map_prefix, 82
- gdsl_bstree_remove, 80
- gdsl_bstree_search, 81
- gdsl_bstree_set_name, 79
- gdsl_bstree_t, 74
- gdsl_bstree_write, 84
- gdsl_bstree_write_xml, 85

Doubly-linked list manipulation module,
127

- gdsl_list_alloc, 130
- gdsl_list_cursor_alloc, 147
- gdsl_list_cursor_delete, 158
- gdsl_list_cursor_delete_after, 158
- gdsl_list_cursor_delete_before, 159
- gdsl_list_cursor_free, 148
- gdsl_list_cursor_get_content, 154
- gdsl_list_cursor_has_pred, 153
- gdsl_list_cursor_has_succ, 152
- gdsl_list_cursor_insert_after, 155
- gdsl_list_cursor_insert_before, 155
- gdsl_list_cursor_is_on_head, 151
- gdsl_list_cursor_is_on_tail, 152
- gdsl_list_cursor_move_to_head, 148
- gdsl_list_cursor_move_to_position,
150
- gdsl_list_cursor_move_to_tail, 149

- gdsl_list_cursor_move_to_value,
149
- gdsl_list_cursor_remove, 156
- gdsl_list_cursor_remove_after, 157
- gdsl_list_cursor_remove_before, 157
- gdsl_list_cursor_set_content, 153
- gdsl_list_cursor_step_backward, 151
- gdsl_list_cursor_step_forward, 150
- gdsl_list_cursor_t, 130
- gdsl_list_delete, 140
- gdsl_list_delete_head, 138
- gdsl_list_delete_tail, 139
- gdsl_list_dump, 146
- gdsl_list_flush, 131
- gdsl_list_free, 131
- gdsl_list_get_head, 133
- gdsl_list_get_name, 132
- gdsl_list_get_size, 132
- gdsl_list_get_tail, 134
- gdsl_list_insert_head, 135
- gdsl_list_insert_tail, 136
- gdsl_list_is_empty, 133
- gdsl_list_map_backward, 144
- gdsl_list_map_forward, 144
- gdsl_list_remove, 138
- gdsl_list_remove_head, 136
- gdsl_list_remove_tail, 137
- gdsl_list_search, 140
- gdsl_list_search_by_position, 141
- gdsl_list_search_max, 142
- gdsl_list_search_min, 142
- gdsl_list_set_name, 134
- gdsl_list_sort, 143
- gdsl_list_t, 130
- gdsl_list_write, 145
- gdsl_list_write_xml, 146

FALSE

GDSDL types, 226

GDSDL types, 222

FALSE, 226

GDSDL_ERR_MEM_ALLOC, 225

- GDSL_FOUND, 225
- GDSL_INSERTED, 225
- GDSL_LOCATION_BOTTOM, 226
- GDSL_LOCATION_FIRST, 226
- GDSL_LOCATION_FIRST_COL, 226
- GDSL_LOCATION_FIRST_ROW, 226
- GDSL_LOCATION_HEAD, 226
- GDSL_LOCATION_LAST, 226
- GDSL_LOCATION_LAST_COL, 226
- GDSL_LOCATION_LAST_ROW, 226
- GDSL_LOCATION_LEAF, 226
- GDSL_LOCATION_ROOT, 226
- GDSL_LOCATION_TAIL, 226
- GDSL_LOCATION_TOP, 226
- GDSL_LOCATION_UNDEF, 226
- GDSL_MAP_CONT, 225
- GDSL_MAP_STOP, 225
- TRUE, 226
- bool, 226
- gdsl_alloc_func_t, 222
- gdsl_compare_func_t, 224
- gdsl_constant_t, 225
- gdsl_copy_func_t, 223
- gdsl_element_t, 222
- gdsl_free_func_t, 223
- gdsl_location_t, 226
- gdsl_map_func_t, 224
- gdsl_write_func_t, 225
- ulong, 225
- ushort, 225
- GDSL_ERR_MEM_ALLOC
 - GDSL types, 225
- GDSL_FOUND
 - GDSL types, 225
- GDSL_INSERTED
 - GDSL types, 225
- GDSL_LOCATION_BOTTOM
 - GDSL types, 226
- GDSL_LOCATION_FIRST
 - GDSL types, 226
- GDSL_LOCATION_FIRST_COL
 - GDSL types, 226
- GDSL_LOCATION_FIRST_ROW
 - GDSL types, 226
- GDSL_LOCATION_HEAD
 - GDSL types, 226
- GDSL_LOCATION_LAST
 - GDSL types, 226
- GDSL_LOCATION_LAST_COL
 - GDSL types, 226
- GDSL_LOCATION_LAST_ROW
 - GDSL types, 226
- GDSL_LOCATION_LEAF
 - GDSL types, 226
- GDSL_LOCATION_ROOT
 - GDSL types, 226
- GDSL_LOCATION_TAIL
 - GDSL types, 226
- GDSL_LOCATION_TOP
 - GDSL types, 226
- GDSL_LOCATION_UNDEF
 - GDSL types, 226
- GDSL_MAP_CONT
 - GDSL types, 225
- GDSL_MAP_STOP
 - GDSL types, 225
- GDSL_MAX
 - Various macros module, 161
- GDSL_MIN
 - Various macros module, 161
- GDSL_PERM_POSITION_FIRST
 - Permutation manipulation module, 165
- GDSL_PERM_POSITION_LAST
 - Permutation manipulation module, 165
- Hashtable manipulation module, 87
 - gdsl_hash, 89
 - gdsl_hash_alloc, 89
 - gdsl_hash_delete, 97
 - gdsl_hash_dump, 101
 - gdsl_hash_flush, 91
 - gdsl_hash_free, 90
 - gdsl_hash_func_t, 88
 - gdsl_hash_get_entries_number, 92
 - gdsl_hash_get_fill_factor, 94
 - gdsl_hash_get_lists_max_size, 93
 - gdsl_hash_get_longest_list_size, 93
 - gdsl_hash_get_name, 91
 - gdsl_hash_get_size, 94
 - gdsl_hash_insert, 96
 - gdsl_hash_map, 99
 - gdsl_hash_modify, 98
 - gdsl_hash_remove, 97
 - gdsl_hash_search, 99
 - gdsl_hash_set_name, 95
 - gdsl_hash_t, 88

- gdsl_hash_write, 100
- gdsl_hash_write_xml, 101
- gdsl_key_func_t, 88
- Heap manipulation module, 103
 - gdsl_heap_alloc, 104
 - gdsl_heap_delete_top, 110
 - gdsl_heap_dump, 113
 - gdsl_heap_flush, 105
 - gdsl_heap_free, 105
 - gdsl_heap_get_name, 106
 - gdsl_heap_get_size, 106
 - gdsl_heap_get_top, 107
 - gdsl_heap_insert, 109
 - gdsl_heap_is_empty, 107
 - gdsl_heap_map_forward, 111
 - gdsl_heap_remove_top, 110
 - gdsl_heap_set_name, 108
 - gdsl_heap_set_top, 108
 - gdsl_heap_t, 104
 - gdsl_heap_write, 112
 - gdsl_heap_write_xml, 112
- Interval Heap manipulation module, 114
 - gdsl_interval_heap_alloc, 115
 - gdsl_interval_heap_delete_max, 123
 - gdsl_interval_heap_delete_min, 123
 - gdsl_interval_heap_dump, 126
 - gdsl_interval_heap_flush, 117
 - gdsl_interval_heap_free, 116
 - gdsl_interval_heap_get_max, 122
 - gdsl_interval_heap_get_min, 122
 - gdsl_interval_heap_get_name, 117
 - gdsl_interval_heap_get_size, 118
 - gdsl_interval_heap_insert, 120
 - gdsl_interval_heap_is_empty, 119
 - gdsl_interval_heap_map_forward, 124
 - gdsl_interval_heap_remove_max, 120
 - gdsl_interval_heap_remove_min, 121
 - gdsl_interval_heap_set_max_size, 118
 - gdsl_interval_heap_set_name, 119
 - gdsl_interval_heap_t, 115
 - gdsl_interval_heap_write, 125
 - gdsl_interval_heap_write_xml, 125
- Low level binary tree manipulation module, 7
 - _gdsl_bintree_alloc, 10
 - _gdsl_bintree_copy, 11
 - _gdsl_bintree_dump, 25
 - _gdsl_bintree_free, 10
 - _gdsl_bintree_get_content, 13
 - _gdsl_bintree_get_height, 16
 - _gdsl_bintree_get_left, 14
 - _gdsl_bintree_get_left_ref, 15
 - _gdsl_bintree_get_parent, 14
 - _gdsl_bintree_get_right, 15
 - _gdsl_bintree_get_right_ref, 16
 - _gdsl_bintree_get_size, 17
 - _gdsl_bintree_is_empty, 11
 - _gdsl_bintree_is_leaf, 12
 - _gdsl_bintree_is_root, 13
 - _gdsl_bintree_map_func_t, 9
 - _gdsl_bintree_map_infix, 23
 - _gdsl_bintree_map_postfix, 23
 - _gdsl_bintree_map_prefix, 22
 - _gdsl_bintree_rotate_left, 20
 - _gdsl_bintree_rotate_left_right, 21
 - _gdsl_bintree_rotate_right, 20
 - _gdsl_bintree_rotate_right_left, 21
 - _gdsl_bintree_set_content, 18
 - _gdsl_bintree_set_left, 19
 - _gdsl_bintree_set_parent, 18
 - _gdsl_bintree_set_right, 19
 - _gdsl_bintree_t, 9
 - _gdsl_bintree_write, 24
 - _gdsl_bintree_write_func_t, 9
 - _gdsl_bintree_write_xml, 25
- Low-level binary search tree manipulation module, 27
 - _gdsl_bstree_alloc, 29
 - _gdsl_bstree_copy, 30
 - _gdsl_bstree_dump, 42
 - _gdsl_bstree_free, 30
 - _gdsl_bstree_get_content, 32
 - _gdsl_bstree_get_height, 35
 - _gdsl_bstree_get_left, 34
 - _gdsl_bstree_get_parent, 33
 - _gdsl_bstree_get_right, 34
 - _gdsl_bstree_get_size, 35
 - _gdsl_bstree_insert, 36
 - _gdsl_bstree_is_empty, 31
 - _gdsl_bstree_is_leaf, 31
 - _gdsl_bstree_is_root, 32
 - _gdsl_bstree_map_func_t, 28
 - _gdsl_bstree_map_infix, 39
 - _gdsl_bstree_map_postfix, 40
 - _gdsl_bstree_map_prefix, 39
 - _gdsl_bstree_remove, 36

- `_gdsl_bstree_search`, 37
- `_gdsl_bstree_search_next`, 38
- `_gdsl_bstree_t`, 28
- `_gdsl_bstree_write`, 41
- `_gdsl_bstree_write_func_t`, 29
- `_gdsl_bstree_write_xml`, 41
- Low-level doubly-linked list manipulation
 - module, 44
 - `_gdsl_list_alloc`, 45
 - `_gdsl_list_dump`, 52
 - `_gdsl_list_free`, 45
 - `_gdsl_list_get_size`, 46
 - `_gdsl_list_insert_after`, 47
 - `_gdsl_list_insert_before`, 48
 - `_gdsl_list_is_empty`, 46
 - `_gdsl_list_link`, 47
 - `_gdsl_list_map_backward`, 50
 - `_gdsl_list_map_forward`, 49
 - `_gdsl_list_remove`, 48
 - `_gdsl_list_search`, 49
 - `_gdsl_list_t`, 45
 - `_gdsl_list_write`, 51
 - `_gdsl_list_write_xml`, 51
- Low-level doubly-linked node manipulation
 - module, 53
 - `_gdsl_node_alloc`, 55
 - `_gdsl_node_dump`, 61
 - `_gdsl_node_free`, 55
 - `_gdsl_node_get_content`, 57
 - `_gdsl_node_get_pred`, 56
 - `_gdsl_node_get_succ`, 55
 - `_gdsl_node_link`, 59
 - `_gdsl_node_map_func_t`, 54
 - `_gdsl_node_set_content`, 58
 - `_gdsl_node_set_pred`, 58
 - `_gdsl_node_set_succ`, 57
 - `_gdsl_node_t`, 54
 - `_gdsl_node_unlink`, 59
 - `_gdsl_node_write`, 60
 - `_gdsl_node_write_func_t`, 54
 - `_gdsl_node_write_xml`, 60
- Main module, 63
 - `gdsl_get_version`, 63
- Permutation manipulation module, 163
 - `GDSL_PERM_POSITION_FIRST`, 165
 - `GDSL_PERM_POSITION_LAST`, 165
 - `gdsl_perm_alloc`, 165
 - `gdsl_perm_apply_on_array`, 177
 - `gdsl_perm_canonical_cycles_count`, 170
 - `gdsl_perm_canonical_to_linear`, 174
 - `gdsl_perm_copy`, 166
 - `gdsl_perm_data_t`, 165
 - `gdsl_perm_dump`, 178
 - `gdsl_perm_free`, 166
 - `gdsl_perm_get_element`, 168
 - `gdsl_perm_get_elements_array`, 169
 - `gdsl_perm_get_name`, 167
 - `gdsl_perm_get_size`, 168
 - `gdsl_perm_inverse`, 175
 - `gdsl_perm_linear_cycles_count`, 170
 - `gdsl_perm_linear_inversions_count`, 169
 - `gdsl_perm_linear_next`, 171
 - `gdsl_perm_linear_prev`, 172
 - `gdsl_perm_linear_to_canonical`, 174
 - `gdsl_perm_multiply`, 173
 - `gdsl_perm_position_t`, 165
 - `gdsl_perm_randomize`, 176
 - `gdsl_perm_reverse`, 176
 - `gdsl_perm_set_elements_array`, 173
 - `gdsl_perm_set_name`, 171
 - `gdsl_perm_t`, 164
 - `gdsl_perm_write`, 177
 - `gdsl_perm_write_func_t`, 164
 - `gdsl_perm_write_xml`, 178
- Queue manipulation module, 180
 - `gdsl_queue_alloc`, 181
 - `gdsl_queue_dump`, 191
 - `gdsl_queue_flush`, 182
 - `gdsl_queue_free`, 182
 - `gdsl_queue_get_head`, 184
 - `gdsl_queue_get_name`, 183
 - `gdsl_queue_get_size`, 183
 - `gdsl_queue_get_tail`, 185
 - `gdsl_queue_insert`, 186
 - `gdsl_queue_is_empty`, 184
 - `gdsl_queue_map_backward`, 189
 - `gdsl_queue_map_forward`, 189
 - `gdsl_queue_remove`, 187
 - `gdsl_queue_search`, 187
 - `gdsl_queue_search_by_position`, 188
 - `gdsl_queue_set_name`, 185
 - `gdsl_queue_t`, 181
 - `gdsl_queue_write`, 190
 - `gdsl_queue_write_xml`, 191
- Red-black tree manipulation module, 193

- gdsl_rbtrees_alloc, 194
- gdsl_rbtrees_delete, 201
- gdsl_rbtrees_dump, 205
- gdsl_rbtrees_flush, 195
- gdsl_rbtrees_free, 195
- gdsl_rbtrees_get_name, 196
- gdsl_rbtrees_get_root, 197
- gdsl_rbtrees_get_size, 197
- gdsl_rbtrees_height, 198
- gdsl_rbtrees_insert, 199
- gdsl_rbtrees_is_empty, 197
- gdsl_rbtrees_map_infix, 203
- gdsl_rbtrees_map_postfix, 203
- gdsl_rbtrees_map_prefix, 202
- gdsl_rbtrees_remove, 200
- gdsl_rbtrees_search, 201
- gdsl_rbtrees_set_name, 199
- gdsl_rbtrees_t, 194
- gdsl_rbtrees_write, 204
- gdsl_rbtrees_write_xml, 205
- Sort module, 207
 - gdsl_sort, 207
- Stack manipulation module, 208
 - gdsl_stack_alloc, 209
 - gdsl_stack_dump, 221
 - gdsl_stack_flush, 210
 - gdsl_stack_free, 210
 - gdsl_stack_get_bottom, 214
 - gdsl_stack_get_growing_factor, 212
 - gdsl_stack_get_name, 211
 - gdsl_stack_get_size, 212
 - gdsl_stack_get_top, 213
 - gdsl_stack_insert, 216
 - gdsl_stack_is_empty, 213
 - gdsl_stack_map_backward, 219
 - gdsl_stack_map_forward, 218
 - gdsl_stack_remove, 216
 - gdsl_stack_search, 217
 - gdsl_stack_search_by_position, 218
 - gdsl_stack_set_growing_factor, 215
 - gdsl_stack_set_name, 214
 - gdsl_stack_t, 209
 - gdsl_stack_write, 219
 - gdsl_stack_write_xml, 220
- TRUE
 - GDSL types, 226
- Various macros module, 161
 - GDSL_MAX, 161
 - GDSL_MIN, 161
- _gdsl_bintree.h, 227
 - _gdsl_bintree_alloc
 - Low level binary tree manipulation module, 10
 - _gdsl_bintree_copy
 - Low level binary tree manipulation module, 11
 - _gdsl_bintree_dump
 - Low level binary tree manipulation module, 25
 - _gdsl_bintree_free
 - Low level binary tree manipulation module, 10
 - _gdsl_bintree_get_content
 - Low level binary tree manipulation module, 13
 - _gdsl_bintree_get_height
 - Low level binary tree manipulation module, 16
 - _gdsl_bintree_get_left
 - Low level binary tree manipulation module, 14
 - _gdsl_bintree_get_left_ref
 - Low level binary tree manipulation module, 15
 - _gdsl_bintree_get_parent
 - Low level binary tree manipulation module, 14
 - _gdsl_bintree_get_right
 - Low level binary tree manipulation module, 15
 - _gdsl_bintree_get_right_ref
 - Low level binary tree manipulation module, 16
 - _gdsl_bintree_get_size
 - Low level binary tree manipulation module, 17
 - _gdsl_bintree_is_empty
 - Low level binary tree manipulation module, 11
 - _gdsl_bintree_is_leaf
 - Low level binary tree manipulation module, 12
 - _gdsl_bintree_is_root
 - Low level binary tree manipulation module, 13
 - _gdsl_bintree_map_func_t
 - Low level binary tree manipulation module, 9
 - _gdsl_bintree_map_infix

- Low level binary tree manipulation module, 23
- `_gdsl_bintree_map_postfix`
 - Low level binary tree manipulation module, 23
- `_gdsl_bintree_map_prefix`
 - Low level binary tree manipulation module, 22
- `_gdsl_bintree_rotate_left`
 - Low level binary tree manipulation module, 20
- `_gdsl_bintree_rotate_left_right`
 - Low level binary tree manipulation module, 21
- `_gdsl_bintree_rotate_right`
 - Low level binary tree manipulation module, 20
- `_gdsl_bintree_rotate_right_left`
 - Low level binary tree manipulation module, 21
- `_gdsl_bintree_set_content`
 - Low level binary tree manipulation module, 18
- `_gdsl_bintree_set_left`
 - Low level binary tree manipulation module, 19
- `_gdsl_bintree_set_parent`
 - Low level binary tree manipulation module, 18
- `_gdsl_bintree_set_right`
 - Low level binary tree manipulation module, 19
- `_gdsl_bintree_t`
 - Low level binary tree manipulation module, 9
- `_gdsl_bintree_write`
 - Low level binary tree manipulation module, 24
- `_gdsl_bintree_write_func_t`
 - Low level binary tree manipulation module, 9
- `_gdsl_bintree_write_xml`
 - Low level binary tree manipulation module, 25
- `_gdsl_bstree.h`, 229
- `_gdsl_bstree_alloc`
 - Low-level binary search tree manipulation module, 29
- `_gdsl_bstree_copy`
 - Low-level binary search tree manipulation module, 30
- `_gdsl_bstree_dump`
 - Low-level binary search tree manipulation module, 42
- `_gdsl_bstree_free`
 - Low-level binary search tree manipulation module, 30
- `_gdsl_bstree_get_content`
 - Low-level binary search tree manipulation module, 32
- `_gdsl_bstree_get_height`
 - Low-level binary search tree manipulation module, 35
- `_gdsl_bstree_get_left`
 - Low-level binary search tree manipulation module, 34
- `_gdsl_bstree_get_parent`
 - Low-level binary search tree manipulation module, 33
- `_gdsl_bstree_get_right`
 - Low-level binary search tree manipulation module, 34
- `_gdsl_bstree_get_size`
 - Low-level binary search tree manipulation module, 35
- `_gdsl_bstree_insert`
 - Low-level binary search tree manipulation module, 36
- `_gdsl_bstree_is_empty`
 - Low-level binary search tree manipulation module, 31
- `_gdsl_bstree_is_leaf`
 - Low-level binary search tree manipulation module, 31
- `_gdsl_bstree_is_root`
 - Low-level binary search tree manipulation module, 32
- `_gdsl_bstree_map_func_t`
 - Low-level binary search tree manipulation module, 28
- `_gdsl_bstree_map_infix`
 - Low-level binary search tree manipulation module, 39
- `_gdsl_bstree_map_postfix`
 - Low-level binary search tree manipulation module, 40
- `_gdsl_bstree_map_prefix`
 - Low-level binary search tree manipulation module, 39

- `_gdsl_bstree_remove`
 - Low-level binary search tree manipulation module, 36
- `_gdsl_bstree_search`
 - Low-level binary search tree manipulation module, 37
- `_gdsl_bstree_search_next`
 - Low-level binary search tree manipulation module, 38
- `_gdsl_bstree_t`
 - Low-level binary search tree manipulation module, 28
- `_gdsl_bstree_write`
 - Low-level binary search tree manipulation module, 41
- `_gdsl_bstree_write_func_t`
 - Low-level binary search tree manipulation module, 29
- `_gdsl_bstree_write_xml`
 - Low-level binary search tree manipulation module, 41
- `_gdsl_list.h`, 230
- `_gdsl_list_alloc`
 - Low-level doubly-linked list manipulation module, 45
- `_gdsl_list_dump`
 - Low-level doubly-linked list manipulation module, 52
- `_gdsl_list_free`
 - Low-level doubly-linked list manipulation module, 45
- `_gdsl_list_get_size`
 - Low-level doubly-linked list manipulation module, 46
- `_gdsl_list_insert_after`
 - Low-level doubly-linked list manipulation module, 47
- `_gdsl_list_insert_before`
 - Low-level doubly-linked list manipulation module, 48
- `_gdsl_list_is_empty`
 - Low-level doubly-linked list manipulation module, 46
- `_gdsl_list_link`
 - Low-level doubly-linked list manipulation module, 47
- `_gdsl_list_map_backward`
 - Low-level doubly-linked list manipulation module, 50
- `_gdsl_list_map_forward`
 - Low-level doubly-linked list manipulation module, 49
- `_gdsl_list_remove`
 - Low-level doubly-linked list manipulation module, 48
- `_gdsl_list_search`
 - Low-level doubly-linked list manipulation module, 49
- `_gdsl_list_t`
 - Low-level doubly-linked list manipulation module, 45
- `_gdsl_list_write`
 - Low-level doubly-linked list manipulation module, 51
- `_gdsl_list_write_xml`
 - Low-level doubly-linked list manipulation module, 51
- `_gdsl_node.h`, 231
- `_gdsl_node_alloc`
 - Low-level doubly-linked node manipulation module, 55
- `_gdsl_node_dump`
 - Low-level doubly-linked node manipulation module, 61
- `_gdsl_node_free`
 - Low-level doubly-linked node manipulation module, 55
- `_gdsl_node_get_content`
 - Low-level doubly-linked node manipulation module, 57
- `_gdsl_node_get_pred`
 - Low-level doubly-linked node manipulation module, 56
- `_gdsl_node_get_succ`
 - Low-level doubly-linked node manipulation module, 55
- `_gdsl_node_link`
 - Low-level doubly-linked node manipulation module, 59
- `_gdsl_node_map_func_t`
 - Low-level doubly-linked node manipulation module, 54
- `_gdsl_node_set_content`
 - Low-level doubly-linked node manipulation module, 58
- `_gdsl_node_set_pred`
 - Low-level doubly-linked node manipulation module, 58
- `_gdsl_node_set_succ`
 - Low-level doubly-linked node manipulation module, 58

- Low-level doubly-linked node manipulation module, 57
- `_gdsl_node_t`
 - Low-level doubly-linked node manipulation module, 54
- `_gdsl_node_unlink`
 - Low-level doubly-linked node manipulation module, 59
- `_gdsl_node_write`
 - Low-level doubly-linked node manipulation module, 60
- `_gdsl_node_write_func_t`
 - Low-level doubly-linked node manipulation module, 54
- `_gdsl_node_write_xml`
 - Low-level doubly-linked node manipulation module, 60
- 2D-Arrays manipulation module, 64
 - `gdsl_2darray_alloc`, 65
 - `gdsl_2darray_dump`, 72
 - `gdsl_2darray_free`, 66
 - `gdsl_2darray_get_columns_number`, 67
 - `gdsl_2darray_get_content`, 68
 - `gdsl_2darray_get_name`, 66
 - `gdsl_2darray_get_rows_number`, 67
 - `gdsl_2darray_get_size`, 68
 - `gdsl_2darray_set_content`, 70
 - `gdsl_2darray_set_name`, 69
 - `gdsl_2darray_t`, 65
 - `gdsl_2darray_write`, 70
 - `gdsl_2darray_write_xml`, 71
- bool
 - GDSL types, 226
- `gdsl.h`, 233
- `gdsl_2darray.h`, 233
- `gdsl_2darray_alloc`
 - 2D-Arrays manipulation module, 65
- `gdsl_2darray_dump`
 - 2D-Arrays manipulation module, 72
- `gdsl_2darray_free`
 - 2D-Arrays manipulation module, 66
- `gdsl_2darray_get_columns_number`
 - 2D-Arrays manipulation module, 67
- `gdsl_2darray_get_content`
 - 2D-Arrays manipulation module, 68
- `gdsl_2darray_get_name`
 - 2D-Arrays manipulation module, 66
- `gdsl_2darray_get_rows_number`
 - 2D-Arrays manipulation module, 67
- `gdsl_2darray_get_size`
 - 2D-Arrays manipulation module, 68
- `gdsl_2darray_set_content`
 - 2D-Arrays manipulation module, 70
- `gdsl_2darray_set_name`
 - 2D-Arrays manipulation module, 69
- `gdsl_2darray_t`
 - 2D-Arrays manipulation module, 65
- `gdsl_2darray_write`
 - 2D-Arrays manipulation module, 70
- `gdsl_2darray_write_xml`
 - 2D-Arrays manipulation module, 71
- `gdsl_alloc_func_t`
 - GDSL types, 222
- `gdsl_bstree.h`, 234
- `gdsl_bstree_alloc`
 - Binary search tree manipulation module, 74
- `gdsl_bstree_delete`
 - Binary search tree manipulation module, 81
- `gdsl_bstree_dump`
 - Binary search tree manipulation module, 86
- `gdsl_bstree_flush`
 - Binary search tree manipulation module, 75
- `gdsl_bstree_free`
 - Binary search tree manipulation module, 75
- `gdsl_bstree_get_height`
 - Binary search tree manipulation module, 78
- `gdsl_bstree_get_name`
 - Binary search tree manipulation module, 76
- `gdsl_bstree_get_root`
 - Binary search tree manipulation module, 77
- `gdsl_bstree_get_size`
 - Binary search tree manipulation module, 78
- `gdsl_bstree_insert`
 - Binary search tree manipulation module, 79
- `gdsl_bstree_is_empty`
 - Binary search tree manipulation module, 77

- gdsl_bstree_map_infix
 - Binary search tree manipulation module, 83
- gdsl_bstree_map_postfix
 - Binary search tree manipulation module, 84
- gdsl_bstree_map_prefix
 - Binary search tree manipulation module, 82
- gdsl_bstree_remove
 - Binary search tree manipulation module, 80
- gdsl_bstree_search
 - Binary search tree manipulation module, 81
- gdsl_bstree_set_name
 - Binary search tree manipulation module, 79
- gdsl_bstree_t
 - Binary search tree manipulation module, 74
- gdsl_bstree_write
 - Binary search tree manipulation module, 84
- gdsl_bstree_write_xml
 - Binary search tree manipulation module, 85
- gdsl_compare_func_t
 - GDSL types, 224
- gdsl_constant_t
 - GDSL types, 225
- gdsl_copy_func_t
 - GDSL types, 223
- gdsl_element_t
 - GDSL types, 222
- gdsl_free_func_t
 - GDSL types, 223
- gdsl_get_version
 - Main module, 63
- gdsl_hash
 - Hashtable manipulation module, 89
- gdsl_hash.h, 235
- gdsl_hash_alloc
 - Hashtable manipulation module, 89
- gdsl_hash_delete
 - Hashtable manipulation module, 97
- gdsl_hash_dump
 - Hashtable manipulation module, 101
- gdsl_hash_flush
 - Hashtable manipulation module, 91
- gdsl_hash_free
 - Hashtable manipulation module, 90
- gdsl_hash_func_t
 - Hashtable manipulation module, 88
- gdsl_hash_get_entries_number
 - Hashtable manipulation module, 92
- gdsl_hash_get_fill_factor
 - Hashtable manipulation module, 94
- gdsl_hash_get_lists_max_size
 - Hashtable manipulation module, 93
- gdsl_hash_get_longest_list_size
 - Hashtable manipulation module, 93
- gdsl_hash_get_name
 - Hashtable manipulation module, 91
- gdsl_hash_get_size
 - Hashtable manipulation module, 94
- gdsl_hash_insert
 - Hashtable manipulation module, 96
- gdsl_hash_map
 - Hashtable manipulation module, 99
- gdsl_hash_modify
 - Hashtable manipulation module, 98
- gdsl_hash_remove
 - Hashtable manipulation module, 97
- gdsl_hash_search
 - Hashtable manipulation module, 99
- gdsl_hash_set_name
 - Hashtable manipulation module, 95
- gdsl_hash_t
 - Hashtable manipulation module, 88
- gdsl_hash_write
 - Hashtable manipulation module, 100
- gdsl_hash_write_xml
 - Hashtable manipulation module, 101
- gdsl_heap.h, 237
- gdsl_heap_alloc
 - Heap manipulation module, 104
- gdsl_heap_delete_top
 - Heap manipulation module, 110
- gdsl_heap_dump
 - Heap manipulation module, 113
- gdsl_heap_flush
 - Heap manipulation module, 105
- gdsl_heap_free
 - Heap manipulation module, 105
- gdsl_heap_get_name
 - Heap manipulation module, 106
- gdsl_heap_get_size
 - Heap manipulation module, 106
- gdsl_heap_get_top

- Heap manipulation module, 107
- gdsl_heap_insert
 - Heap manipulation module, 109
- gdsl_heap_is_empty
 - Heap manipulation module, 107
- gdsl_heap_map_forward
 - Heap manipulation module, 111
- gdsl_heap_remove_top
 - Heap manipulation module, 110
- gdsl_heap_set_name
 - Heap manipulation module, 108
- gdsl_heap_set_top
 - Heap manipulation module, 108
- gdsl_heap_t
 - Heap manipulation module, 104
- gdsl_heap_write
 - Heap manipulation module, 112
- gdsl_heap_write_xml
 - Heap manipulation module, 112
- gdsl_interval_heap.h, 238
- gdsl_interval_heap_alloc
 - Interval Heap manipulation module, 115
- gdsl_interval_heap_delete_max
 - Interval Heap manipulation module, 123
- gdsl_interval_heap_delete_min
 - Interval Heap manipulation module, 123
- gdsl_interval_heap_dump
 - Interval Heap manipulation module, 126
- gdsl_interval_heap_flush
 - Interval Heap manipulation module, 117
- gdsl_interval_heap_free
 - Interval Heap manipulation module, 116
- gdsl_interval_heap_get_max
 - Interval Heap manipulation module, 122
- gdsl_interval_heap_get_min
 - Interval Heap manipulation module, 122
- gdsl_interval_heap_get_name
 - Interval Heap manipulation module, 117
- gdsl_interval_heap_get_size
 - Interval Heap manipulation module, 118
- gdsl_interval_heap_insert
 - Interval Heap manipulation module, 120
- gdsl_interval_heap_is_empty
 - Interval Heap manipulation module, 119
- gdsl_interval_heap_map_forward
 - Interval Heap manipulation module, 124
- gdsl_interval_heap_remove_max
 - Interval Heap manipulation module, 120
- gdsl_interval_heap_remove_min
 - Interval Heap manipulation module, 121
- gdsl_interval_heap_set_max_size
 - Interval Heap manipulation module, 118
- gdsl_interval_heap_set_name
 - Interval Heap manipulation module, 119
- gdsl_interval_heap_t
 - Interval Heap manipulation module, 115
- gdsl_interval_heap_write
 - Interval Heap manipulation module, 125
- gdsl_interval_heap_write_xml
 - Interval Heap manipulation module, 125
- gdsl_key_func_t
 - Hashtable manipulation module, 88
- gdsl_list.h, 239
- gdsl_list_alloc
 - Doubly-linked list manipulation module, 130
- gdsl_list_cursor_alloc
 - Doubly-linked list manipulation module, 147
- gdsl_list_cursor_delete
 - Doubly-linked list manipulation module, 158
- gdsl_list_cursor_delete_after
 - Doubly-linked list manipulation module, 158
- gdsl_list_cursor_delete_before
 - Doubly-linked list manipulation module, 159
- gdsl_list_cursor_free

- Doubly-linked list manipulation module, 148
- gdsl_list_cursor_get_content
 - Doubly-linked list manipulation module, 154
- gdsl_list_cursor_has_pred
 - Doubly-linked list manipulation module, 153
- gdsl_list_cursor_has_succ
 - Doubly-linked list manipulation module, 152
- gdsl_list_cursor_insert_after
 - Doubly-linked list manipulation module, 155
- gdsl_list_cursor_insert_before
 - Doubly-linked list manipulation module, 155
- gdsl_list_cursor_is_on_head
 - Doubly-linked list manipulation module, 151
- gdsl_list_cursor_is_on_tail
 - Doubly-linked list manipulation module, 152
- gdsl_list_cursor_move_to_head
 - Doubly-linked list manipulation module, 148
- gdsl_list_cursor_move_to_position
 - Doubly-linked list manipulation module, 150
- gdsl_list_cursor_move_to_tail
 - Doubly-linked list manipulation module, 149
- gdsl_list_cursor_move_to_value
 - Doubly-linked list manipulation module, 149
- gdsl_list_cursor_remove
 - Doubly-linked list manipulation module, 156
- gdsl_list_cursor_remove_after
 - Doubly-linked list manipulation module, 157
- gdsl_list_cursor_remove_before
 - Doubly-linked list manipulation module, 157
- gdsl_list_cursor_set_content
 - Doubly-linked list manipulation module, 153
- gdsl_list_cursor_step_backward
 - Doubly-linked list manipulation module, 151
- gdsl_list_cursor_step_forward
 - Doubly-linked list manipulation module, 150
- gdsl_list_cursor_t
 - Doubly-linked list manipulation module, 130
- gdsl_list_delete
 - Doubly-linked list manipulation module, 140
- gdsl_list_delete_head
 - Doubly-linked list manipulation module, 138
- gdsl_list_delete_tail
 - Doubly-linked list manipulation module, 139
- gdsl_list_dump
 - Doubly-linked list manipulation module, 146
- gdsl_list_flush
 - Doubly-linked list manipulation module, 131
- gdsl_list_free
 - Doubly-linked list manipulation module, 131
- gdsl_list_get_head
 - Doubly-linked list manipulation module, 133
- gdsl_list_get_name
 - Doubly-linked list manipulation module, 132
- gdsl_list_get_size
 - Doubly-linked list manipulation module, 132
- gdsl_list_get_tail
 - Doubly-linked list manipulation module, 134
- gdsl_list_insert_head
 - Doubly-linked list manipulation module, 135
- gdsl_list_insert_tail
 - Doubly-linked list manipulation module, 136
- gdsl_list_is_empty
 - Doubly-linked list manipulation module, 133
- gdsl_list_map_backward
 - Doubly-linked list manipulation module, 144
- gdsl_list_map_forward

- Doubly-linked list manipulation module, 144
- gdsl_list_remove
 - Doubly-linked list manipulation module, 138
- gdsl_list_remove_head
 - Doubly-linked list manipulation module, 136
- gdsl_list_remove_tail
 - Doubly-linked list manipulation module, 137
- gdsl_list_search
 - Doubly-linked list manipulation module, 140
- gdsl_list_search_by_position
 - Doubly-linked list manipulation module, 141
- gdsl_list_search_max
 - Doubly-linked list manipulation module, 142
- gdsl_list_search_min
 - Doubly-linked list manipulation module, 142
- gdsl_list_set_name
 - Doubly-linked list manipulation module, 134
- gdsl_list_sort
 - Doubly-linked list manipulation module, 143
- gdsl_list_t
 - Doubly-linked list manipulation module, 130
- gdsl_list_write
 - Doubly-linked list manipulation module, 145
- gdsl_list_write_xml
 - Doubly-linked list manipulation module, 146
- gdsl_location_t
 - GDSL types, 226
- gdsl_macros.h, 242
- gdsl_map_func_t
 - GDSL types, 224
- gdsl_perm.h, 242
- gdsl_perm_alloc
 - Permutation manipulation module, 165
- gdsl_perm_apply_on_array
 - Permutation manipulation module, 177
- gdsl_perm_canonical_cycles_count
 - Permutation manipulation module, 170
- gdsl_perm_canonical_to_linear
 - Permutation manipulation module, 174
- gdsl_perm_copy
 - Permutation manipulation module, 166
- gdsl_perm_data_t
 - Permutation manipulation module, 165
- gdsl_perm_dump
 - Permutation manipulation module, 178
- gdsl_perm_free
 - Permutation manipulation module, 166
- gdsl_perm_get_element
 - Permutation manipulation module, 168
- gdsl_perm_get_elements_array
 - Permutation manipulation module, 169
- gdsl_perm_get_name
 - Permutation manipulation module, 167
- gdsl_perm_get_size
 - Permutation manipulation module, 168
- gdsl_perm_inverse
 - Permutation manipulation module, 175
- gdsl_perm_linear_cycles_count
 - Permutation manipulation module, 170
- gdsl_perm_linear_inversions_count
 - Permutation manipulation module, 169
- gdsl_perm_linear_next
 - Permutation manipulation module, 171
- gdsl_perm_linear_prev
 - Permutation manipulation module, 172
- gdsl_perm_linear_to_canonical
 - Permutation manipulation module, 174
- gdsl_perm_multiply

- Permutation manipulation module, 173
- gdsI_perm_position_t
 - Permutation manipulation module, 165
- gdsI_perm_randomize
 - Permutation manipulation module, 176
- gdsI_perm_reverse
 - Permutation manipulation module, 176
- gdsI_perm_set_elements_array
 - Permutation manipulation module, 173
- gdsI_perm_set_name
 - Permutation manipulation module, 171
- gdsI_perm_t
 - Permutation manipulation module, 164
- gdsI_perm_write
 - Permutation manipulation module, 177
- gdsI_perm_write_func_t
 - Permutation manipulation module, 164
- gdsI_perm_write_xml
 - Permutation manipulation module, 178
- gdsI_queue.h, 244
- gdsI_queue_alloc
 - Queue manipulation module, 181
- gdsI_queue_dump
 - Queue manipulation module, 191
- gdsI_queue_flush
 - Queue manipulation module, 182
- gdsI_queue_free
 - Queue manipulation module, 182
- gdsI_queue_get_head
 - Queue manipulation module, 184
- gdsI_queue_get_name
 - Queue manipulation module, 183
- gdsI_queue_get_size
 - Queue manipulation module, 183
- gdsI_queue_get_tail
 - Queue manipulation module, 185
- gdsI_queue_insert
 - Queue manipulation module, 186
- gdsI_queue_is_empty
 - Queue manipulation module, 184
- gdsI_queue_map_backward
 - Queue manipulation module, 189
- gdsI_queue_map_forward
 - Queue manipulation module, 189
- gdsI_queue_remove
 - Queue manipulation module, 187
- gdsI_queue_search
 - Queue manipulation module, 187
- gdsI_queue_search_by_position
 - Queue manipulation module, 188
- gdsI_queue_set_name
 - Queue manipulation module, 185
- gdsI_queue_t
 - Queue manipulation module, 181
- gdsI_queue_write
 - Queue manipulation module, 190
- gdsI_queue_write_xml
 - Queue manipulation module, 191
- gdsI_rbtrees.h, 245
- gdsI_rbtrees_alloc
 - Red-black tree manipulation module, 194
- gdsI_rbtrees_delete
 - Red-black tree manipulation module, 201
- gdsI_rbtrees_dump
 - Red-black tree manipulation module, 205
- gdsI_rbtrees_flush
 - Red-black tree manipulation module, 195
- gdsI_rbtrees_free
 - Red-black tree manipulation module, 195
- gdsI_rbtrees_get_name
 - Red-black tree manipulation module, 196
- gdsI_rbtrees_get_root
 - Red-black tree manipulation module, 197
- gdsI_rbtrees_get_size
 - Red-black tree manipulation module, 197
- gdsI_rbtrees_height
 - Red-black tree manipulation module, 198
- gdsI_rbtrees_insert
 - Red-black tree manipulation module, 199
- gdsI_rbtrees_is_empty

- Red-black tree manipulation module, 197
- gdsl_rbtrees_map_infix
 - Red-black tree manipulation module, 203
- gdsl_rbtrees_map_postfix
 - Red-black tree manipulation module, 203
- gdsl_rbtrees_map_prefix
 - Red-black tree manipulation module, 202
- gdsl_rbtrees_remove
 - Red-black tree manipulation module, 200
- gdsl_rbtrees_search
 - Red-black tree manipulation module, 201
- gdsl_rbtrees_set_name
 - Red-black tree manipulation module, 199
- gdsl_rbtrees_t
 - Red-black tree manipulation module, 194
- gdsl_rbtrees_write
 - Red-black tree manipulation module, 204
- gdsl_rbtrees_write_xml
 - Red-black tree manipulation module, 205
- gdsl_sort
 - Sort module, 207
- gdsl_sort.h, 247
- gdsl_stack.h, 247
- gdsl_stack_alloc
 - Stack manipulation module, 209
- gdsl_stack_dump
 - Stack manipulation module, 221
- gdsl_stack_flush
 - Stack manipulation module, 210
- gdsl_stack_free
 - Stack manipulation module, 210
- gdsl_stack_get_bottom
 - Stack manipulation module, 214
- gdsl_stack_get_growing_factor
 - Stack manipulation module, 212
- gdsl_stack_get_name
 - Stack manipulation module, 211
- gdsl_stack_get_size
 - Stack manipulation module, 212
- gdsl_stack_get_top
 - Stack manipulation module, 213
- gdsl_stack_insert
 - Stack manipulation module, 216
- gdsl_stack_is_empty
 - Stack manipulation module, 213
- gdsl_stack_map_backward
 - Stack manipulation module, 219
- gdsl_stack_map_forward
 - Stack manipulation module, 218
- gdsl_stack_remove
 - Stack manipulation module, 216
- gdsl_stack_search
 - Stack manipulation module, 217
- gdsl_stack_search_by_position
 - Stack manipulation module, 218
- gdsl_stack_set_growing_factor
 - Stack manipulation module, 215
- gdsl_stack_set_name
 - Stack manipulation module, 214
- gdsl_stack_t
 - Stack manipulation module, 209
- gdsl_stack_write
 - Stack manipulation module, 219
- gdsl_stack_write_xml
 - Stack manipulation module, 220
- gdsl_types.h, 248
- gdsl_write_func_t
 - GDSL types, 225
- mainpage.h, 249
- ulong
 - GDSL types, 225
- ushort
 - GDSL types, 225