

gdsI  
1.8

Generated by Doxygen 1.7.6.1

Mon Sep 25 2017 18:08:14



# Contents

<b>1</b>	<b>gdsl</b>	<b>1</b>
1.1	Introduction . . . . .	1
1.2	About . . . . .	1
1.3	Copyright . . . . .	1
1.4	Authors . . . . .	2
1.5	Project Manager . . . . .	2
1.6	Thanks . . . . .	2
<b>2</b>	<b>Module Index</b>	<b>5</b>
2.1	Modules . . . . .	5
<b>3</b>	<b>File Index</b>	<b>7</b>
3.1	File List . . . . .	7
<b>4</b>	<b>Module Documentation</b>	<b>9</b>
4.1	Low level binary tree manipulation module . . . . .	9
4.1.1	. . . . .	9
4.1.2	Copyright . . . . .	9
4.1.3	Typedef Documentation . . . . .	11
4.1.3.1	_gdsl_bintree_t . . . . .	11
4.1.3.2	_gdsl_bintree_map_func_t . . . . .	11
4.1.3.3	_gdsl_bintree_write_func_t . . . . .	12
4.1.4	Function Documentation . . . . .	12
4.1.4.1	_gdsl_bintree_alloc . . . . .	12
4.1.4.2	_gdsl_bintree_free . . . . .	13
4.1.4.3	_gdsl_bintree_copy . . . . .	13

4.1.4.4	<code>_gdsl_bintree_is_empty</code> . . . . .	14
4.1.4.5	<code>_gdsl_bintree_is_leaf</code> . . . . .	15
4.1.4.6	<code>_gdsl_bintree_is_root</code> . . . . .	15
4.1.4.7	<code>_gdsl_bintree_get_content</code> . . . . .	16
4.1.4.8	<code>_gdsl_bintree_get_parent</code> . . . . .	17
4.1.4.9	<code>_gdsl_bintree_get_left</code> . . . . .	17
4.1.4.10	<code>_gdsl_bintree_get_right</code> . . . . .	18
4.1.4.11	<code>_gdsl_bintree_get_left_ref</code> . . . . .	18
4.1.4.12	<code>_gdsl_bintree_get_right_ref</code> . . . . .	19
4.1.4.13	<code>_gdsl_bintree_get_height</code> . . . . .	19
4.1.4.14	<code>_gdsl_bintree_get_size</code> . . . . .	20
4.1.4.15	<code>_gdsl_bintree_set_content</code> . . . . .	21
4.1.4.16	<code>_gdsl_bintree_set_parent</code> . . . . .	21
4.1.4.17	<code>_gdsl_bintree_set_left</code> . . . . .	22
4.1.4.18	<code>_gdsl_bintree_set_right</code> . . . . .	22
4.1.4.19	<code>_gdsl_bintree_rotate_left</code> . . . . .	23
4.1.4.20	<code>_gdsl_bintree_rotate_right</code> . . . . .	23
4.1.4.21	<code>_gdsl_bintree_rotate_left_right</code> . . . . .	24
4.1.4.22	<code>_gdsl_bintree_rotate_right_left</code> . . . . .	25
4.1.4.23	<code>_gdsl_bintree_map_prefix</code> . . . . .	25
4.1.4.24	<code>_gdsl_bintree_map_infix</code> . . . . .	26
4.1.4.25	<code>_gdsl_bintree_map_postfix</code> . . . . .	27
4.1.4.26	<code>_gdsl_bintree_write</code> . . . . .	28
4.1.4.27	<code>_gdsl_bintree_write_xml</code> . . . . .	28
4.1.4.28	<code>_gdsl_bintree_dump</code> . . . . .	29
4.2	Low-level binary search tree manipulation module . . . . .	31
4.2.1	. . . . .	31
4.2.2	Copyright . . . . .	31
4.2.3	Typedef Documentation . . . . .	33
4.2.3.1	<code>_gdsl_bstree_t</code> . . . . .	33
4.2.3.2	<code>_gdsl_bstree_map_func_t</code> . . . . .	33
4.2.3.3	<code>_gdsl_bstree_write_func_t</code> . . . . .	33
4.2.4	Function Documentation . . . . .	33
4.2.4.1	<code>_gdsl_bstree_alloc</code> . . . . .	34

4.2.4.2	_gdsl_bstree_free . . . . .	34
4.2.4.3	_gdsl_bstree_copy . . . . .	35
4.2.4.4	_gdsl_bstree_is_empty . . . . .	35
4.2.4.5	_gdsl_bstree_is_leaf . . . . .	36
4.2.4.6	_gdsl_bstree_get_content . . . . .	37
4.2.4.7	_gdsl_bstree_is_root . . . . .	37
4.2.4.8	_gdsl_bstree_get_parent . . . . .	38
4.2.4.9	_gdsl_bstree_get_left . . . . .	38
4.2.4.10	_gdsl_bstree_get_right . . . . .	39
4.2.4.11	_gdsl_bstree_get_size . . . . .	39
4.2.4.12	_gdsl_bstree_get_height . . . . .	40
4.2.4.13	_gdsl_bstree_insert . . . . .	40
4.2.4.14	_gdsl_bstree_remove . . . . .	41
4.2.4.15	_gdsl_bstree_search . . . . .	42
4.2.4.16	_gdsl_bstree_search_next . . . . .	43
4.2.4.17	_gdsl_bstree_map_prefix . . . . .	43
4.2.4.18	_gdsl_bstree_map_infix . . . . .	44
4.2.4.19	_gdsl_bstree_map_postfix . . . . .	45
4.2.4.20	_gdsl_bstree_write . . . . .	46
4.2.4.21	_gdsl_bstree_write_xml . . . . .	46
4.2.4.22	_gdsl_bstree_dump . . . . .	47
4.3	Low-level doubly-linked list manipulation module . . . . .	48
4.3.1	. . . . .	48
4.3.2	Copyright . . . . .	48
4.3.3	Typedef Documentation . . . . .	49
4.3.3.1	_gdsl_list_t . . . . .	49
4.3.4	Function Documentation . . . . .	49
4.3.4.1	_gdsl_list_alloc . . . . .	49
4.3.4.2	_gdsl_list_free . . . . .	50
4.3.4.3	_gdsl_list_is_empty . . . . .	50
4.3.4.4	_gdsl_list_get_size . . . . .	51
4.3.4.5	_gdsl_list_link . . . . .	51
4.3.4.6	_gdsl_list_insert_after . . . . .	52
4.3.4.7	_gdsl_list_insert_before . . . . .	52

4.3.4.8	<code>_gdsl_list_remove</code> . . . . .	53
4.3.4.9	<code>_gdsl_list_search</code> . . . . .	53
4.3.4.10	<code>_gdsl_list_map_forward</code> . . . . .	54
4.3.4.11	<code>_gdsl_list_map_backward</code> . . . . .	55
4.3.4.12	<code>_gdsl_list_write</code> . . . . .	55
4.3.4.13	<code>_gdsl_list_write_xml</code> . . . . .	56
4.3.4.14	<code>_gdsl_list_dump</code> . . . . .	57
4.4	Low-level doubly-linked node manipulation module . . . . .	59
4.4.1	. . . . .	59
4.4.2	Copyright . . . . .	59
4.4.3	Typedef Documentation . . . . .	60
4.4.3.1	<code>_gdsl_node_t</code> . . . . .	60
4.4.3.2	<code>_gdsl_node_map_func_t</code> . . . . .	60
4.4.3.3	<code>_gdsl_node_write_func_t</code> . . . . .	61
4.4.4	Function Documentation . . . . .	61
4.4.4.1	<code>_gdsl_node_alloc</code> . . . . .	61
4.4.4.2	<code>_gdsl_node_free</code> . . . . .	61
4.4.4.3	<code>_gdsl_node_get_succ</code> . . . . .	62
4.4.4.4	<code>_gdsl_node_get_pred</code> . . . . .	62
4.4.4.5	<code>_gdsl_node_get_content</code> . . . . .	63
4.4.4.6	<code>_gdsl_node_set_succ</code> . . . . .	64
4.4.4.7	<code>_gdsl_node_set_pred</code> . . . . .	64
4.4.4.8	<code>_gdsl_node_set_content</code> . . . . .	65
4.4.4.9	<code>_gdsl_node_link</code> . . . . .	65
4.4.4.10	<code>_gdsl_node_unlink</code> . . . . .	66
4.4.4.11	<code>_gdsl_node_write</code> . . . . .	66
4.4.4.12	<code>_gdsl_node_write_xml</code> . . . . .	67
4.4.4.13	<code>_gdsl_node_dump</code> . . . . .	68
4.5	Main module . . . . .	69
4.5.1	. . . . .	69
4.5.2	Copyright . . . . .	69
4.5.3	Function Documentation . . . . .	69
4.5.3.1	<code>gdsl_get_version</code> . . . . .	69
4.6	2D-Arrays manipulation module . . . . .	70

---

4.6.1	.....	70
4.6.2	Copyright .....	70
4.6.3	Typedef Documentation .....	71
4.6.3.1	gdsl_2darray_t .....	71
4.6.4	Function Documentation .....	71
4.6.4.1	gdsl_2darray_alloc .....	71
4.6.4.2	gdsl_2darray_free .....	72
4.6.4.3	gdsl_2darray_get_name .....	72
4.6.4.4	gdsl_2darray_get_rows_number .....	73
4.6.4.5	gdsl_2darray_get_columns_number .....	74
4.6.4.6	gdsl_2darray_get_size .....	74
4.6.4.7	gdsl_2darray_get_content .....	75
4.6.4.8	gdsl_2darray_set_name .....	75
4.6.4.9	gdsl_2darray_set_content .....	76
4.6.4.10	gdsl_2darray_write .....	77
4.6.4.11	gdsl_2darray_write_xml .....	77
4.6.4.12	gdsl_2darray_dump .....	78
4.7	Binary search tree manipulation module .....	80
4.7.1	.....	80
4.7.2	Copyright .....	80
4.7.3	Typedef Documentation .....	81
4.7.3.1	gdsl_bstree_t .....	81
4.7.4	Function Documentation .....	81
4.7.4.1	gdsl_bstree_alloc .....	81
4.7.4.2	gdsl_bstree_free .....	82
4.7.4.3	gdsl_bstree_flush .....	83
4.7.4.4	gdsl_bstree_get_name .....	84
4.7.4.5	gdsl_bstree_is_empty .....	84
4.7.4.6	gdsl_bstree_get_root .....	85
4.7.4.7	gdsl_bstree_get_size .....	85
4.7.4.8	gdsl_bstree_get_height .....	86
4.7.4.9	gdsl_bstree_set_name .....	86
4.7.4.10	gdsl_bstree_insert .....	87
4.7.4.11	gdsl_bstree_remove .....	88

---

4.7.4.12	gdsl_bstree_delete . . . . .	89
4.7.4.13	gdsl_bstree_search . . . . .	89
4.7.4.14	gdsl_bstree_map_prefix . . . . .	90
4.7.4.15	gdsl_bstree_map_infix . . . . .	91
4.7.4.16	gdsl_bstree_map_postfix . . . . .	92
4.7.4.17	gdsl_bstree_write . . . . .	92
4.7.4.18	gdsl_bstree_write_xml . . . . .	93
4.7.4.19	gdsl_bstree_dump . . . . .	94
4.8	Hashtable manipulation module . . . . .	95
4.8.1	. . . . .	95
4.8.2	Copyright . . . . .	95
4.8.3	Typedef Documentation . . . . .	96
4.8.3.1	gdsl_hash_t . . . . .	96
4.8.3.2	gdsl_key_func_t . . . . .	97
4.8.3.3	gdsl_hash_func_t . . . . .	97
4.8.4	Function Documentation . . . . .	97
4.8.4.1	gdsl_hash . . . . .	97
4.8.4.2	gdsl_hash_alloc . . . . .	98
4.8.4.3	gdsl_hash_free . . . . .	99
4.8.4.4	gdsl_hash_flush . . . . .	99
4.8.4.5	gdsl_hash_get_name . . . . .	100
4.8.4.6	gdsl_hash_get_entries_number . . . . .	101
4.8.4.7	gdsl_hash_get_lists_max_size . . . . .	101
4.8.4.8	gdsl_hash_get_longest_list_size . . . . .	102
4.8.4.9	gdsl_hash_get_size . . . . .	102
4.8.4.10	gdsl_hash_get_fill_factor . . . . .	103
4.8.4.11	gdsl_hash_set_name . . . . .	104
4.8.4.12	gdsl_hash_insert . . . . .	104
4.8.4.13	gdsl_hash_remove . . . . .	105
4.8.4.14	gdsl_hash_delete . . . . .	106
4.8.4.15	gdsl_hash_modify . . . . .	107
4.8.4.16	gdsl_hash_search . . . . .	108
4.8.4.17	gdsl_hash_map . . . . .	108
4.8.4.18	gdsl_hash_write . . . . .	109



4.8.4.19	gdsl_hash_write_xml . . . . .	110
4.8.4.20	gdsl_hash_dump . . . . .	110
4.9	Heap manipulation module . . . . .	112
4.9.1	. . . . .	112
4.9.2	Copyright . . . . .	112
4.9.3	Typedef Documentation . . . . .	113
4.9.3.1	gdsl_heap_t . . . . .	113
4.9.4	Function Documentation . . . . .	113
4.9.4.1	gdsl_heap_alloc . . . . .	113
4.9.4.2	gdsl_heap_free . . . . .	114
4.9.4.3	gdsl_heap_flush . . . . .	115
4.9.4.4	gdsl_heap_get_name . . . . .	115
4.9.4.5	gdsl_heap_get_size . . . . .	116
4.9.4.6	gdsl_heap_get_top . . . . .	116
4.9.4.7	gdsl_heap_is_empty . . . . .	117
4.9.4.8	gdsl_heap_set_name . . . . .	118
4.9.4.9	gdsl_heap_set_top . . . . .	118
4.9.4.10	gdsl_heap_insert . . . . .	119
4.9.4.11	gdsl_heap_remove_top . . . . .	120
4.9.4.12	gdsl_heap_delete_top . . . . .	120
4.9.4.13	gdsl_heap_map_forward . . . . .	121
4.9.4.14	gdsl_heap_write . . . . .	122
4.9.4.15	gdsl_heap_write_xml . . . . .	122
4.9.4.16	gdsl_heap_dump . . . . .	123
4.10	Interval Heap manipulation module . . . . .	125
4.10.1	. . . . .	125
4.10.2	Copyright . . . . .	125
4.10.3	Typedef Documentation . . . . .	126
4.10.3.1	gdsl_interval_heap_t . . . . .	126
4.10.4	Function Documentation . . . . .	127
4.10.4.1	gdsl_interval_heap_alloc . . . . .	127
4.10.4.2	gdsl_interval_heap_free . . . . .	128
4.10.4.3	gdsl_interval_heap_flush . . . . .	128
4.10.4.4	gdsl_interval_heap_get_name . . . . .	129

4.10.4.5	gdsl_interval_heap_get_size . . . . .	129
4.10.4.6	gdsl_interval_heap_set_max_size . . . . .	130
4.10.4.7	gdsl_interval_heap_is_empty . . . . .	130
4.10.4.8	gdsl_interval_heap_set_name . . . . .	131
4.10.4.9	gdsl_interval_heap_insert . . . . .	131
4.10.4.10	gdsl_interval_heap_remove_max . . . . .	132
4.10.4.11	gdsl_interval_heap_remove_min . . . . .	133
4.10.4.12	gdsl_interval_heap_get_min . . . . .	134
4.10.4.13	gdsl_interval_heap_get_max . . . . .	134
4.10.4.14	gdsl_interval_heap_delete_min . . . . .	135
4.10.4.15	gdsl_interval_heap_delete_max . . . . .	135
4.10.4.16	gdsl_interval_heap_map_forward . . . . .	136
4.10.4.17	gdsl_interval_heap_write . . . . .	136
4.10.4.18	gdsl_interval_heap_write_xml . . . . .	137
4.10.4.19	gdsl_interval_heap_dump . . . . .	138
4.11	Doubly-linked list manipulation module . . . . .	139
4.11.1	. . . . .	139
4.11.2	Copyright . . . . .	139
4.11.3	Typedef Documentation . . . . .	142
4.11.3.1	gdsl_list_t . . . . .	142
4.11.3.2	gdsl_list_cursor_t . . . . .	142
4.11.4	Function Documentation . . . . .	142
4.11.4.1	gdsl_list_alloc . . . . .	142
4.11.4.2	gdsl_list_free . . . . .	143
4.11.4.3	gdsl_list_flush . . . . .	144
4.11.4.4	gdsl_list_get_name . . . . .	144
4.11.4.5	gdsl_list_get_size . . . . .	145
4.11.4.6	gdsl_list_is_empty . . . . .	145
4.11.4.7	gdsl_list_get_head . . . . .	146
4.11.4.8	gdsl_list_get_tail . . . . .	147
4.11.4.9	gdsl_list_set_name . . . . .	147
4.11.4.10	gdsl_list_insert_head . . . . .	148
4.11.4.11	gdsl_list_insert_tail . . . . .	149
4.11.4.12	gdsl_list_remove_head . . . . .	149

4.11.4.13 gdslist_remove_tail . . . . .	150
4.11.4.14 gdslist_remove . . . . .	151
4.11.4.15 gdslist_delete_head . . . . .	151
4.11.4.16 gdslist_delete_tail . . . . .	152
4.11.4.17 gdslist_delete . . . . .	153
4.11.4.18 gdslist_search . . . . .	154
4.11.4.19 gdslist_search_by_position . . . . .	154
4.11.4.20 gdslist_search_max . . . . .	155
4.11.4.21 gdslist_search_min . . . . .	156
4.11.4.22 gdslist_sort . . . . .	157
4.11.4.23 gdslist_map_forward . . . . .	157
4.11.4.24 gdslist_map_backward . . . . .	158
4.11.4.25 gdslist_write . . . . .	159
4.11.4.26 gdslist_write_xml . . . . .	159
4.11.4.27 gdslist_dump . . . . .	160
4.11.4.28 gdslist_cursor_alloc . . . . .	161
4.11.4.29 gdslist_cursor_free . . . . .	161
4.11.4.30 gdslist_cursor_move_to_head . . . . .	162
4.11.4.31 gdslist_cursor_move_to_tail . . . . .	162
4.11.4.32 gdslist_cursor_move_to_value . . . . .	163
4.11.4.33 gdslist_cursor_move_to_position . . . . .	164
4.11.4.34 gdslist_cursor_step_forward . . . . .	164
4.11.4.35 gdslist_cursor_step_backward . . . . .	165
4.11.4.36 gdslist_cursor_is_on_head . . . . .	165
4.11.4.37 gdslist_cursor_is_on_tail . . . . .	166
4.11.4.38 gdslist_cursor_has_succ . . . . .	167
4.11.4.39 gdslist_cursor_has_pred . . . . .	167
4.11.4.40 gdslist_cursor_set_content . . . . .	168
4.11.4.41 gdslist_cursor_get_content . . . . .	168
4.11.4.42 gdslist_cursor_insert_after . . . . .	169
4.11.4.43 gdslist_cursor_insert_before . . . . .	170
4.11.4.44 gdslist_cursor_remove . . . . .	170
4.11.4.45 gdslist_cursor_remove_after . . . . .	171
4.11.4.46 gdslist_cursor_remove_before . . . . .	172

4.11.4.47	gdsl_list_cursor_delete . . . . .	172
4.11.4.48	gdsl_list_cursor_delete_after . . . . .	173
4.11.4.49	gdsl_list_cursor_delete_before . . . . .	173
4.12	Various macros module . . . . .	175
4.12.1	. . . . .	175
4.12.2	Copyright . . . . .	175
4.12.3	Define Documentation . . . . .	175
4.12.3.1	GDSL_MAX . . . . .	175
4.12.3.2	GDSL_MIN . . . . .	176
4.13	Permutation manipulation module . . . . .	177
4.13.1	. . . . .	177
4.13.2	Copyright . . . . .	177
4.13.3	Typedef Documentation . . . . .	179
4.13.3.1	gdsl_perm_t . . . . .	179
4.13.3.2	gdsl_perm_write_func_t . . . . .	179
4.13.3.3	gdsl_perm_data_t . . . . .	179
4.13.4	Enumeration Type Documentation . . . . .	179
4.13.4.1	gdsl_perm_position_t . . . . .	179
4.13.5	Function Documentation . . . . .	180
4.13.5.1	gdsl_perm_alloc . . . . .	180
4.13.5.2	gdsl_perm_free . . . . .	180
4.13.5.3	gdsl_perm_copy . . . . .	181
4.13.5.4	gdsl_perm_get_name . . . . .	182
4.13.5.5	gdsl_perm_get_size . . . . .	182
4.13.5.6	gdsl_perm_get_element . . . . .	183
4.13.5.7	gdsl_perm_get_elements_array . . . . .	183
4.13.5.8	gdsl_perm_linear_inversions_count . . . . .	184
4.13.5.9	gdsl_perm_linear_cycles_count . . . . .	184
4.13.5.10	gdsl_perm_canonical_cycles_count . . . . .	185
4.13.5.11	gdsl_perm_set_name . . . . .	186
4.13.5.12	gdsl_perm_linear_next . . . . .	186
4.13.5.13	gdsl_perm_linear_prev . . . . .	187
4.13.5.14	gdsl_perm_set_elements_array . . . . .	188
4.13.5.15	gdsl_perm_multiply . . . . .	188

4.13.5.16	gdsl_perm_linear_to_canonical . . . . .	189
4.13.5.17	gdsl_perm_canonical_to_linear . . . . .	189
4.13.5.18	gdsl_perm_inverse . . . . .	190
4.13.5.19	gdsl_perm_reverse . . . . .	191
4.13.5.20	gdsl_perm_randomize . . . . .	191
4.13.5.21	gdsl_perm_apply_on_array . . . . .	192
4.13.5.22	gdsl_perm_write . . . . .	193
4.13.5.23	gdsl_perm_write_xml . . . . .	193
4.13.5.24	gdsl_perm_dump . . . . .	194
4.14	Queue manipulation module . . . . .	195
4.14.1	. . . . .	195
4.14.2	Copyright . . . . .	195
4.14.3	Typedef Documentation . . . . .	196
4.14.3.1	gdsl_queue_t . . . . .	196
4.14.4	Function Documentation . . . . .	196
4.14.4.1	gdsl_queue_alloc . . . . .	196
4.14.4.2	gdsl_queue_free . . . . .	197
4.14.4.3	gdsl_queue_flush . . . . .	198
4.14.4.4	gdsl_queue_get_name . . . . .	198
4.14.4.5	gdsl_queue_get_size . . . . .	199
4.14.4.6	gdsl_queue_is_empty . . . . .	200
4.14.4.7	gdsl_queue_get_head . . . . .	200
4.14.4.8	gdsl_queue_get_tail . . . . .	201
4.14.4.9	gdsl_queue_set_name . . . . .	201
4.14.4.10	gdsl_queue_insert . . . . .	202
4.14.4.11	gdsl_queue_remove . . . . .	203
4.14.4.12	gdsl_queue_search . . . . .	203
4.14.4.13	gdsl_queue_search_by_position . . . . .	204
4.14.4.14	gdsl_queue_map_forward . . . . .	205
4.14.4.15	gdsl_queue_map_backward . . . . .	206
4.14.4.16	gdsl_queue_write . . . . .	206
4.14.4.17	gdsl_queue_write_xml . . . . .	207
4.14.4.18	gdsl_queue_dump . . . . .	208
4.15	Red-black tree manipulation module . . . . .	209

4.15.1	209
4.15.2 Copyright	209
4.15.3 Typedef Documentation	210
4.15.3.1 gdsl_rbtrees_t	210
4.15.4 Function Documentation	210
4.15.4.1 gdsl_rbtrees_alloc	210
4.15.4.2 gdsl_rbtrees_free	211
4.15.4.3 gdsl_rbtrees_flush	212
4.15.4.4 gdsl_rbtrees_get_name	213
4.15.4.5 gdsl_rbtrees_is_empty	213
4.15.4.6 gdsl_rbtrees_get_root	214
4.15.4.7 gdsl_rbtrees_get_size	214
4.15.4.8 gdsl_rbtrees_height	215
4.15.4.9 gdsl_rbtrees_set_name	215
4.15.4.10 gdsl_rbtrees_insert	216
4.15.4.11 gdsl_rbtrees_remove	217
4.15.4.12 gdsl_rbtrees_delete	218
4.15.4.13 gdsl_rbtrees_search	218
4.15.4.14 gdsl_rbtrees_map_prefix	219
4.15.4.15 gdsl_rbtrees_map_infix	220
4.15.4.16 gdsl_rbtrees_map_postfix	221
4.15.4.17 gdsl_rbtrees_write	221
4.15.4.18 gdsl_rbtrees_write_xml	222
4.15.4.19 gdsl_rbtrees_dump	223
4.16 Sort module	225
4.16.1	225
4.16.2 Copyright	225
4.16.3 Function Documentation	225
4.16.3.1 gdsl_sort	225
4.17 Stack manipulation module	227
4.17.1	227
4.17.2 Copyright	227
4.17.3 Typedef Documentation	228
4.17.3.1 gdsl_stack_t	228

4.17.4	Function Documentation . . . . .	229
4.17.4.1	gdsl_stack_alloc . . . . .	229
4.17.4.2	gdsl_stack_free . . . . .	229
4.17.4.3	gdsl_stack_flush . . . . .	230
4.17.4.4	gdsl_stack_get_name . . . . .	231
4.17.4.5	gdsl_stack_get_size . . . . .	231
4.17.4.6	gdsl_stack_get_growing_factor . . . . .	232
4.17.4.7	gdsl_stack_is_empty . . . . .	232
4.17.4.8	gdsl_stack_get_top . . . . .	233
4.17.4.9	gdsl_stack_get_bottom . . . . .	233
4.17.4.10	gdsl_stack_set_name . . . . .	234
4.17.4.11	gdsl_stack_set_growing_factor . . . . .	235
4.17.4.12	gdsl_stack_insert . . . . .	235
4.17.4.13	gdsl_stack_remove . . . . .	236
4.17.4.14	gdsl_stack_search . . . . .	237
4.17.4.15	gdsl_stack_search_by_position . . . . .	237
4.17.4.16	gdsl_stack_map_forward . . . . .	238
4.17.4.17	gdsl_stack_map_backward . . . . .	239
4.17.4.18	gdsl_stack_write . . . . .	239
4.17.4.19	gdsl_stack_write_xml . . . . .	240
4.17.4.20	gdsl_stack_dump . . . . .	241
4.18	GDSL types . . . . .	242
4.18.1	. . . . .	242
4.18.2	Copyright . . . . .	242
4.18.3	Typedef Documentation . . . . .	243
4.18.3.1	gdsl_element_t . . . . .	243
4.18.3.2	gdsl_alloc_func_t . . . . .	243
4.18.3.3	gdsl_free_func_t . . . . .	243
4.18.3.4	gdsl_copy_func_t . . . . .	244
4.18.3.5	gdsl_map_func_t . . . . .	244
4.18.3.6	gdsl_compare_func_t . . . . .	245
4.18.3.7	gdsl_write_func_t . . . . .	245
4.18.3.8	ulong . . . . .	245
4.18.3.9	ushort . . . . .	246

4.18.4	Enumeration Type Documentation . . . . .	246
4.18.4.1	gdsl_constant_t . . . . .	246
4.18.4.2	gdsl_location_t . . . . .	246
4.18.4.3	bool . . . . .	247
<b>5</b>	<b>File Documentation</b>	<b>249</b>
5.1	_gdsl_bintree.h File Reference . . . . .	249
5.2	_gdsl_bstree.h File Reference . . . . .	251
5.3	_gdsl_list.h File Reference . . . . .	252
5.4	_gdsl_node.h File Reference . . . . .	253
5.5	gdsl.h File Reference . . . . .	255
5.6	gdsl_2darray.h File Reference . . . . .	255
5.7	gdsl_bstree.h File Reference . . . . .	256
5.8	gdsl_hash.h File Reference . . . . .	257
5.9	gdsl_heap.h File Reference . . . . .	259
5.10	gdsl_interval_heap.h File Reference . . . . .	260
5.11	gdsl_list.h File Reference . . . . .	261
5.12	gdsl_macros.h File Reference . . . . .	264
5.13	gdsl_perm.h File Reference . . . . .	264
5.14	gdsl_queue.h File Reference . . . . .	266
5.15	gdsl_rbtrees.h File Reference . . . . .	267
5.16	gdsl_sort.h File Reference . . . . .	269
5.17	gdsl_stack.h File Reference . . . . .	269
5.18	gdsl_types.h File Reference . . . . .	270
5.19	mainpage.h File Reference . . . . .	271
<b>6</b>	<b>Example Documentation</b>	<b>273</b>
6.1	examples/main_bstree.c . . . . .	273
6.2	examples/main_hash.c . . . . .	276
6.3	examples/main_heap.c . . . . .	280
6.4	examples/main_interval_heap.c . . . . .	283
6.5	examples/main_list.c . . . . .	287
6.6	examples/main_llbintree.c . . . . .	295
6.7	examples/main_llbstree.c . . . . .	296
6.8	examples/main_lllist.c . . . . .	299



6.9	examples/main_perm.c . . . . .	301
6.10	examples/main_queue.c . . . . .	303
6.11	examples/main_rbtrees.c . . . . .	307
6.12	examples/main_sort.c . . . . .	310
6.13	examples/main_stack.c . . . . .	312



# Chapter 1

## gdsl

### 1.1 Introduction

This is the gdsl (Release 1.8) 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, heaps and interval heaps.

### 1.3 Copyright

Copyright (C) 1998-2017 Nicolas Darnis <ndarnis@free.fr>.

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

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

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

## 1.4 Authors

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

Peter Kerpedjiev <pkerpedjiev@gmail.com>: interval\_heap module.

## 1.5 Project Manager

Nicolas Darnis <ndarnis@free.fr>.

## 1.6 Thanks

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

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

For his bug report in hash\_insert method and into **gdsl.h** (p. 255).

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

For his patch to compile GDSDL under OSX.

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

For his bug report in **gdsl\_stack\_insert()** (p. 235).

- Xavier De Labouret <Xavier.de\_Labouret@cvf.fr>

For his bug report in **gdsl\_hash\_search()** (p. 108).

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

For his KazLib from wich the deletion algorithm for gdsl\_rbtrees.c is inspired.

- David Lewin <dlewin@free.fr>

For his bug report in **gdsl\_list\_map\_backward()** (p. 158), and for the problem of re-defining bool type in **gdsl\_types.h** (p. 270).

- Torsten Luetzgert <t.luetzgert@combox.de>

For his gdsI.spec file to build GDSDL's RPM package.

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

For his patch to compile GDSDL under FreeBSD.

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

For his patch to compile GDSDL under Android OS.

- Peter Kerpedjiev <pkerpedjiev@gmail.com>

For his gdsI\_interval\_heap module.

- Benny Pasternak <bennypk>

For his is bug report in gdsI\_rbtrees\_map\_infix function.

The GDSDL Team.



## Chapter 2

# Module Index

### 2.1 Modules

Here is a list of all modules:

Low level binary tree manipulation module . . . . .	9
Low-level binary search tree manipulation module . . . . .	31
Low-level doubly-linked list manipulation module . . . . .	48
Low-level doubly-linked node manipulation module . . . . .	59
Main module . . . . .	69
2D-Arrays manipulation module . . . . .	70
Binary search tree manipulation module . . . . .	80
Hashtable manipulation module . . . . .	95
Heap manipulation module . . . . .	112
Interval Heap manipulation module . . . . .	125
Doubly-linked list manipulation module . . . . .	139
Various macros module . . . . .	175
Permutation manipulation module . . . . .	177
Queue manipulation module . . . . .	195
Red-black tree manipulation module . . . . .	209
Sort module . . . . .	225
Stack manipulation module . . . . .	227
GDSL types . . . . .	242





## Chapter 3

# File Index

### 3.1 File List

Here is a list of all files with brief descriptions:

<code>_gdsi_bintree.h</code>	249
<code>_gdsi_bstree.h</code>	251
<code>_gdsi_list.h</code>	252
<code>_gdsi_node.h</code>	253
<code>gdsi.h</code>	255
<code>gdsi_2darray.h</code>	255
<code>gdsi_bstree.h</code>	256
<code>gdsi_hash.h</code>	257
<code>gdsi_heap.h</code>	259
<code>gdsi_interval_heap.h</code>	260
<code>gdsi_list.h</code>	261
<code>gdsi_macros.h</code>	264
<code>gdsi_perm.h</code>	264
<code>gdsi_queue.h</code>	266
<code>gdsi_rbtrees.h</code>	267
<code>gdsi_sort.h</code>	269
<code>gdsi_stack.h</code>	269
<code>gdsi_types.h</code>	270
<code>mainpage.h</code>	271



## Chapter 4

# Module Documentation

### 4.1 Low level binary tree manipulation module

#### 4.1.1

#### 4.1.2 Copyright

Copyright (C) 1998-2017 Nicolas Darnis <ndarnis@free.fr>.

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

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

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

### 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

- **\_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.*
- **gdsI\_element\_t \_gdsI\_bintree\_get\_content** (const **\_gdsI\_bintree\_t** T)  
*Get the root content of a low-level binary tree.*
- **\_gdsI\_bintree\_t \_gdsI\_bintree\_get\_parent** (const **\_gdsI\_bintree\_t** T)  
*Get the parent tree of a low-level binary tree.*
- **\_gdsI\_bintree\_t \_gdsI\_bintree\_get\_left** (const **\_gdsI\_bintree\_t** T)  
*Get the left sub-tree of a low-level binary tree.*
- **\_gdsI\_bintree\_t \_gdsI\_bintree\_get\_right** (const **\_gdsI\_bintree\_t** T)  
*Get the right sub-tree of a low-level binary tree.*
- **\_gdsI\_bintree\_t \* \_gdsI\_bintree\_get\_left\_ref** (const **\_gdsI\_bintree\_t** T)  
*Get the left sub-tree reference of a low-level binary tree.*
- **\_gdsI\_bintree\_t \* \_gdsI\_bintree\_get\_right\_ref** (const **\_gdsI\_bintree\_t** T)  
*Get the right sub-tree reference of a low-level binary tree.*
- **ulong \_gdsI\_bintree\_get\_height** (const **\_gdsI\_bintree\_t** T)  
*Get the height of a low-level binary tree.*
- **ulong \_gdsI\_bintree\_get\_size** (const **\_gdsI\_bintree\_t** T)  
*Get the size of a low-level binary tree.*
- **void \_gdsI\_bintree\_set\_content** (**\_gdsI\_bintree\_t** T, const **gdsI\_element\_t** E)  
*Set the root element of a low-level binary tree.*
- **void \_gdsI\_bintree\_set\_parent** (**\_gdsI\_bintree\_t** T, const **\_gdsI\_bintree\_t** P)  
*Set the parent tree of a low-level binary tree.*
- **void \_gdsI\_bintree\_set\_left** (**\_gdsI\_bintree\_t** T, const **\_gdsI\_bintree\_t** L)  
*Set left sub-tree of a low-level binary tree.*
- **void \_gdsI\_bintree\_set\_right** (**\_gdsI\_bintree\_t** T, const **\_gdsI\_bintree\_t** R)  
*Set right sub-tree of a low-level binary tree.*
- **\_gdsI\_bintree\_t \_gdsI\_bintree\_rotate\_left** (**\_gdsI\_bintree\_t** \*T)  
*Left rotate a low-level binary tree.*

- **`_gdsl_bintree_t _gsdl_bintree_rotate_right (_gsdl_bintree_t *T)`**  
*Right rotate a low-level binary tree.*
- **`_gdsl_bintree_t _gsdl_bintree_rotate_left_right (_gsdl_bintree_t *T)`**  
*Left-right rotate a low-level binary tree.*
- **`_gdsl_bintree_t _gsdl_bintree_rotate_right_left (_gsdl_bintree_t *T)`**  
*Right-left rotate a low-level binary tree.*
- **`_gsdl_bintree_t _gsdl_bintree_map_prefix (const _gsdl_bintree_t T, const _gsdl_bintree_map_func_t MAP_F, void *USER_DATA)`**  
*Parse a low-level binary tree in prefixed order.*
- **`_gsdl_bintree_t _gsdl_bintree_map_infix (const _gsdl_bintree_t T, const _gsdl_bintree_map_func_t MAP_F, void *USER_DATA)`**  
*Parse a low-level binary tree in infix order.*
- **`_gsdl_bintree_t _gsdl_bintree_map_postfix (const _gsdl_bintree_t T, const _gsdl_bintree_map_func_t MAP_F, void *USER_DATA)`**  
*Parse a low-level binary tree in postfix order.*
- **`void _gsdl_bintree_write (const _gsdl_bintree_t T, const _gsdl_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 _gsdl_bintree_write_xml (const _gsdl_bintree_t T, const _gsdl_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 _gsdl_bintree_dump (const _gsdl_bintree_t T, const _gsdl_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.*

### 4.1.3 Typedef Documentation

#### 4.1.3.1 typedef struct \_gsdl\_bintree\* \_gsdl\_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 73 of file `_gsdl_bintree.h`.

#### 4.1.3.2 typedef int(\* \_gsdl\_bintree\_map\_func\_t)(const \_gsdl\_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 82 of file \_gdsl\_bintree.h.

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

GDSL low-level binary tree write function type.

**Parameters**

<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 92 of file \_gdsl\_bintree.h.

**4.1.4 Function Documentation**

**4.1.4.1** `_gdsl_bintree_t _gdsl_bintree_alloc( const gdsl_element_t E, const _gdsl_bintree_t LEFT, const _gdsl_bintree_t RIGHT )`

Create a new low-level binary tree.

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

**Note**

Complexity:  $O(1)$

**Precondition**

nothing.

**Parameters**

<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**

**`_gdsI_bintree_free()`** (p. 13)

**Examples:**

**`examples/main_llbintree.c`.**

4.1.4.2 `void _gdsI_bintree_free( _gdsI_bintree_t T, const gdsI_free_func_t FREE_F )`

Destroy a low-level binary tree.

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

**Note**

Complexity:  $O(|T|)$

**Precondition**

nothing.

**Parameters**

<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. 12)

**Examples:**

**`examples/main_llbintree.c`.**

4.1.4.3 `_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**

<code>T</code>	The low-level binary tree to copy.
<code>COPY_F</code>	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. 12)  
`_gdsi_bintree_free()` (p. 13)  
`_gdsi_bintree_is_empty()` (p. 14)

**Examples:**

`examples/main_llbintree.c.`

**4.1.4.4 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**

[`\_gdsl\_bintree\_is\_leaf\(\)`](#) (p. 15)  
[`\_gdsl\_bintree\_is\_root\(\)`](#) (p. 15)

**4.1.4.5 `bool _gdsl_bintree_is_leaf( const _gdsl_bintree_t T )`**

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

**Note**

Complexity:  $O(1)$

**Precondition**

T must be a non-empty `_gdsl_bintree_t`.

**Parameters**

<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**

[`\_gdsl\_bintree\_is\_empty\(\)`](#) (p. 14)  
[`\_gdsl\_bintree\_is\_root\(\)`](#) (p. 15)

**4.1.4.6 `bool _gdsl_bintree_is_root( const _gdsl_bintree_t T )`**

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

**Note**

Complexity:  $O(1)$

**Precondition**

T must be a non-empty `_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. 14)  
`_gdsI_bintree_is_leaf()` (p. 15)

**4.1.4.7 `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. 21)

**Examples:**

`examples/main_llbintree.c`.

#### 4.1.4.8 `_gdsl_bintree_t _gdsl_bintree_get_parent( const _gdsl_bintree_t T )`

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

##### Note

Complexity:  $O(1)$

##### Precondition

T must be a non-empty `_gdsl_bintree_t`.

##### Parameters

<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

`_gdsl_bintree_is_root()` (p. 15)

`_gdsl_bintree_set_parent()` (p. 21)

#### 4.1.4.9 `_gdsl_bintree_t _gdsl_bintree_get_left( const _gdsl_bintree_t T )`

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

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

##### Note

Complexity:  $O(1)$

##### Precondition

T must be a non-empty `_gdsl_bintree_t`.

##### Parameters

<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**

`_gdsI_bintree_get_right()` (p. 18)  
`_gdsI_bintree_set_left()` (p. 22)  
`_gdsI_bintree_set_right()` (p. 22)

**4.1.4.10 `_gdsI_bintree_t _gdsI_bintree_get_right( const _gdsI_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 `_gdsI_bintree_t`

**Parameters**

<i>T</i>	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**

`_gdsI_bintree_get_left()` (p. 17)  
`_gdsI_bintree_set_left()` (p. 22)  
`_gdsI_bintree_set_right()` (p. 22)

**4.1.4.11 `_gdsI_bintree_t* _gdsI_bintree_get_left_ref( const _gdsI_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 `_gdsi_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**

`_gdsi_bintree_get_right_ref()` (p. 19)

**4.1.4.12 `_gdsi_bintree_t* _gdsi_bintree_get_right_ref( const _gdsi_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 `_gdsi_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**

`_gdsi_bintree_get_left_ref()` (p. 18)

**4.1.4.13 `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. 20)

**4.1.4.14 `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. 19)

4.1.4.15 `void _gdsl_bintree_set_content( _gdsl_bintree_t T, const gdsl_element_t E )`

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

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

**Note**

Complexity:  $O(1)$

**Precondition**

T must be a non-empty `_gdsl_bintree_t`.

**Parameters**

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

**See also**

`_gdsl_bintree_get_content` (p. 16)

4.1.4.16 `void _gdsl_bintree_set_parent( _gdsl_bintree_t T, const _gdsl_bintree_t P )`

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

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

**Note**

Complexity:  $O(1)$

**Precondition**

T must be a non-empty `_gdsl_bintree_t`.

**Parameters**

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

**See also**

`_gdsl_bintree_get_parent()` (p. 17)

4.1.4.17 `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. 22)

`_gdsi_bintree_get_left()` (p. 17)

`_gdsi_bintree_get_right()` (p. 18)

4.1.4.18 `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

`_gdsl_bintree_set_left()` (p. 22)  
`_gdsl_bintree_get_left()` (p. 17)  
`_gdsl_bintree_get_right()` (p. 18)

#### 4.1.4.19 `_gsdl_bintree_t _gsdl_bintree_rotate_left( _gsdl_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 `_gsdl_bintree_t`.

Parameters

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

Returns

the modified T left-rotated.

See also

`_gsdl_bintree_rotate_right()` (p. 23)  
`_gsdl_bintree_rotate_left_right()` (p. 24)  
`_gsdl_bintree_rotate_right_left()` (p. 25)

Examples:

**examples/main\_llbintree.c.**

#### 4.1.4.20 `_gsdl_bintree_t _gsdl_bintree_rotate_right( _gsdl_bintree_t * T )`

Right rotate a low-level binary tree.

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

Note

Complexity:  $O(1)$

**Precondition**

$T$  &  $l(T)$  must be non-empty `_gdsl_bintree_t`.

**Parameters**

$T$	The low-level binary tree to rotate.
-----	--------------------------------------

**Returns**

the modified  $T$  right-rotated.

**See also**

`_gdsl_bintree_rotate_left()` (p. 23)  
`_gdsl_bintree_rotate_left_right()` (p. 24)  
`_gdsl_bintree_rotate_right_left()` (p. 25)

**Examples:**

`examples/main_llbintree.c`.

**4.1.4.21 `_gdsl_bintree_t _gdsl_bintree_rotate_left_right( _gdsl_bintree_t *  $T$  )`**

Left-right rotate a low-level binary tree.

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

**Note**

Complexity:  $O(1)$

**Precondition**

$T$  &  $l(T)$  &  $r(l(T))$  must be non-empty `_gdsl_bintree_t`.

**Parameters**

$T$	The low-level binary tree to rotate.
-----	--------------------------------------

**Returns**

the modified  $T$  left-right-rotated.

See also

`_gdsl_bintree_rotate_left()` (p. 23)  
`_gdsl_bintree_rotate_right()` (p. 23)  
`_gdsl_bintree_rotate_right_left()` (p. 25)

#### 4.1.4.22 `_gdsl_bintree_t _gdsl_bintree_rotate_right_left( _gdsl_bintree_t * T )`

Right-left rotate a low-level binary tree.

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

Note

Complexity:  $O(1)$

Precondition

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

Parameters

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

Returns

the modified T right-left-rotated.

See also

`_gdsl_bintree_rotate_left()` (p. 23)  
`_gdsl_bintree_rotate_right()` (p. 23)  
`_gdsl_bintree_rotate_left_right()` (p. 24)

#### 4.1.4.23 `_gdsl_bintree_t _gdsl_bintree_map_prefix( const _gdsl_bintree_t T, const _gdsl_bintree_map_func_t MAP_F, void * USER_DATA )`

Parse a low-level binary tree in prefixed order.

Parse all nodes of the low-level binary tree T in prefixed order. The MAP\_F function is called on each node with the USER\_DATA argument. If MAP\_F returns GDSL\_MAP\_STOP, then `_gdsl_bintree_map_prefix()` (p. 25) stops and returns its last 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 GD\_SL\_MAP\_STOP.  
NULL when the parsing is done.

**See also**

[\\_gdsl\\_bintree\\_map\\_infix\(\)](#) (p. 26)  
[\\_gdsl\\_bintree\\_map\\_postfix\(\)](#) (p. 27)

**Examples:**

[examples/main\\_llbintree.c](#).

#### 4.1.4.24 `_gdsl_bintree_t _gdsl_bintree_map_infix( const _gdsl_bintree_t T, const _gdsl_bintree_map_func_t MAP_F, void * USER_DATA )`

Parse a low-level binary tree in 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 GD\_SL\_MAP\_STOP, then [\\_gdsl\\_bintree\\_map\\_infix\(\)](#) (p. 26) 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. 25)  
`_gdsl_bintree_map_postfix()` (p. 27)

**Examples:**

`examples/main_llbintree.c.`

4.1.4.25 `_gdsl_bintree_t _gdsl_bintree_map_postfix( const _gdsl_bintree_t T,  
 const _gdsl_bintree_map_func_t MAP_F, void * USER_DATA )`

Parse a low-level binary tree in 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. 27) 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. 25)  
`_gdsl_bintree_map_infix()` (p. 26)

Examples:

`examples/main_llbintree.c.`

4.1.4.26 `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.

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. 28)

`_gdsi_bintree_dump()` (p. 29)

4.1.4.27 `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. 28)

`_gdsi_bintree_dump()` (p. 29)

## Examples:

`examples/main_llbintree.c`.

4.1.4.28 `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.

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 T's nodes.
<i>USER_DATA</i>	User's datas passed to WRITE_F.

See also

`_gdsi_bintree_write()` (p. 28)

`_gdsi_bintree_write_xml()` (p. 28)

Examples:

`examples/main_llbintree.c`.



## 4.2 Low-level binary search tree manipulation module

### 4.2.1

### 4.2.2 Copyright

Copyright (C) 1998-2017 Nicolas Darnis <ndarnis@free.fr>.

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

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

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

### 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 \_gsdl\_bstree\_get\_parent** (const \_gsdl\_bstree\_t T)

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

- **\_gsdl\_bstree\_t \_gsdl\_bstree\_get\_left** (const \_gsdl\_bstree\_t T)

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

- **\_gsdl\_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.*

### 4.2.3 Typedef Documentation

#### 4.2.3.1 typedef `_gdsi_bintree_t` `_gdsi_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 71 of file `_gdsi_bstree.h`.

#### 4.2.3.2 typedef `int(* _gdsi_bstree_map_func_t)(_gdsi_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>	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 80 of file `_gdsi_bstree.h`.

#### 4.2.3.3 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.

##### 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>	The user datas to pass to this function.

Definition at line 90 of file `_gdsi_bstree.h`.

### 4.2.4 Function Documentation

#### 4.2.4.1 `_gdsi_bstree_t _gdsi_bstree_alloc( const gdsi_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

<code>E</code>	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

`_gdsi_bstree_free()` (p. 34)

##### Examples:

`examples/main_llbstree.c`.

#### 4.2.4.2 `void _gdsi_bstree_free( _gdsi_bstree_t T, const gdsi_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. 34)

Examples:

[`examples/main\_llbstree.c`](#).

#### 4.2.4.3 `_gdsl_bstree_t _gdsl_bstree_copy( const _gdsl_bstree_t T, const gdsl_copy_func_t COPY_F )`

Copy a low-level binary search tree.

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

Note

Complexity:  $O(|T|)$

Precondition

`COPY_F != NULL`.

Parameters

<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 `_gdsl_bstree_is_empty(T) == TRUE` or in case of insufficient memory.

See also

[`\_gdsl\_bstree\_alloc\(\)`](#) (p. 34)

[`\_gdsl\_bstree\_free\(\)`](#) (p. 34)

[`\_gdsl\_bstree\_is\_empty\(\)`](#) (p. 35)

#### 4.2.4.4 `bool _gdsl_bstree_is_empty( const _gdsl_bstree_t T )`

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

**Note**

Complexity:  $O(1)$

**Precondition**

nothing.

**Parameters**

<b><i>T</i></b>	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. 36)  
[`\_gdsi\_bstree\_is\_root\(\)`](#) (p. 37)

**4.2.4.5 `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**

<b><i>T</i></b>	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. 35)  
[`\_gdsi\_bstree\_is\_root\(\)`](#) (p. 37)

#### 4.2.4.6 `gdsl_element_t _gdsl_bstree_get_content( const _gdsl_bstree_t T )`

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

##### Note

Complexity:  $O(1)$

##### Precondition

T must be a non-empty `_gdsl_bstree_t`.

##### Parameters

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

##### Returns

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

##### Examples:

**examples/main\_llbstree.c.**

#### 4.2.4.7 `bool _gdsl_bstree_is_root( const _gdsl_bstree_t T )`

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

##### Note

Complexity:  $O(1)$

##### Precondition

T must be a non-empty `_gdsl_bstree_t`.

##### Parameters

<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. 35)

`_gdsI_bstree_is_leaf()` (p. 36)

#### 4.2.4.8 `_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. 37)

#### 4.2.4.9 `_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. 39)

**4.2.4.10 `_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**

<i>T</i>	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. 38)

**4.2.4.11 `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

<code>T</code>	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. 40)

#### 4.2.4.12 `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

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

## Returns

the height of `T`.

## See also

`_gdsi_bstree_get_size()` (p. 39)

#### 4.2.4.13 `_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.

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 T's elements with VALUE to find E.
<i>VALUE</i>	The value used to search for the element E.
<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. 42)  
[\\_gdsl\\_bstree\\_remove\(\)](#) (p. 41)

**Examples:**

[examples/main\\_lbstree.c](#).

#### 4.2.4.14 `gdsl_element_t gdsl_bstree_remove( _gdsl_bstree_t * T, const gdsl_compare_func_t COMP_F, const gdsl_element_t VALUE )`

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

Remove from the low-level binary search tree T the first founded element E equal to 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. 40)  
`_gdsi_bstree_search()` (p. 42)

#### 4.2.4.15 `_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

`_gdsl_bstree_insert()` (p. 40)  
`_gdsl_bstree_remove()` (p. 41)

4.2.4.16 `_gdsl_bstree_t _gdsl_bstree_search_next( const _gdsl_bstree_t T, const  
 _gdsl_compare_func_t COMP_F, const _gdsl_element_t VALUE )`

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

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

Note

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

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 successor.

4.2.4.17 `_gdsl_bstree_t _gdsl_bstree_map_prefix( const _gdsl_bstree_t T, const  
 _gdsl_bstree_map_func_t MAP_F, void * USER_DATA )`

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

Parse all nodes of the low-level binary search tree T in prefixed order. The MAP\_F function is called on each node with the USER\_DATA argument. If MAP\_F returns -GDSL\_MAP\_STOP, then `_gdsl_bstree_map_prefix()` (p. 43) stops and returns its last 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**

[\\_gdsl\\_bstree\\_map\\_infix\(\)](#) (p. 44)  
[\\_gdsl\\_bstree\\_map\\_postfix\(\)](#) (p. 45)

#### 4.2.4.18 `_gdsl_bstree_t _gdsl_bstree_map_infix( const _gdsl_bstree_t T, const _gdsl_bstree_map_func_t MAP_F, void * USER_DATA )`

Parse a low-level binary search tree in 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 [\\_gdsl\\_bstree\\_map\\_infix\(\)](#) (p. 44) 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 GDSL\_MAP\_STOP.  
 NULL when the parsing is done.

**See also**

`_gdsl_bstree_map_prefix()` (p. 43)  
`_gdsl_bstree_map_postfix()` (p. 45)

4.2.4.19 `_gdsl_bstree_t _gdsl_bstree_map_postfix( const _gdsl_bstree_t T, const  
 _gdsl_bstree_map_func_t MAP_F, void * USER_DATA )`

Parse a low-level binary search tree in 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 `_gdsl_bstree_map_postfix()` (p. 45) 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 GDSL\_MAP\_STOP.  
 NULL when the parsing is done.

**See also**

`_gdsl_bstree_map_prefix()` (p. 43)  
`_gdsl_bstree_map_infix()` (p. 44)

4.2.4.20 `void _gdsi_bstree_write( const _gdsi_bstree_t T, const  
_gdsi_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

`_gdsi_bstree_write_xml()` (p. 46)

`_gdsi_bstree_dump()` (p. 47)

4.2.4.21 `void _gdsi_bstree_write_xml( const _gdsi_bstree_t T, const  
_gdsi_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

[`\_gdsl\_bstree\_write\(\)`](#) (p. 46)

[`\_gdsl\_bstree\_dump\(\)`](#) (p. 47)

## Examples:

`examples/main_llbstree.c`.

4.2.4.22 `void _gdsl_bstree_dump ( const _gdsl_bstree_t T, const  
_gdsl_bstree_write_func_t WRITE_F, FILE * OUTPUT_FILE, void * USER_DATA )`

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

Dump the structure of the low-level binary search tree T to OUTPUT\_FILE. If WRITE\_F != NULL, then use WRITE\_F function to write T's nodes content to OUTPUT\_FILE. Additionnal USER\_DATA argument could be passed to WRITE\_F.

## Note

Complexity:  $O(|T|)$

## Precondition

OUTPUT\_FILE != NULL.

## Parameters

<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.

## See also

[`\_gdsl\_bstree\_write\(\)`](#) (p. 46)

[`\_gdsl\_bstree\_write\_xml\(\)`](#) (p. 46)

## 4.3 Low-level doubly-linked list manipulation module

### 4.3.1

### 4.3.2 Copyright

Copyright (C) 1998-2017 Nicolas Darnis <ndarnis@free.fr>.

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

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

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

### 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.*

- **`_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.*

### 4.3.3 Typedef Documentation

#### 4.3.3.1 typedef **`_gdsi_node_t _gdsi_list_t`**

GDSL low-level doubly-linked list type.

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

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

### 4.3.4 Function Documentation

#### 4.3.4.1 **`_gdsi_list_t _gdsi_list_alloc( const gdsi_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

<b><code>E</code></b>	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**

**`_gdsI_list_free()`** (p. 50)

**Examples:**

**`examples/main_llist.c`.**

#### 4.3.4.2 void `_gdsI_list_free( _gdsI_list_t L, const gdsI_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**

**`_gdsI_list_alloc()`** (p. 49)

**Examples:**

**`examples/main_llist.c`.**

#### 4.3.4.3 bool `_gdsI_list_is_empty( const _gdsI_list_t L )`

Check if a low-level list is empty.

**Note**

Complexity:  $O(1)$

**Precondition**

nothing.

**Parameters**

$L$	The low-level list to check.
-----	------------------------------

**Returns**

TRUE if the low-level list  $L$  is empty.

FALSE if the low-level list  $L$  is not empty.

**4.3.4.4 `ulong _gdsi_list_get_size( const _gdsi_list_t L )`**

Get the size of a low-level list.

**Note**

Complexity:  $O(|L|)$

**Precondition**

nothing.

**Parameters**

$L$	The low-level list to use.
-----	----------------------------

**Returns**

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

**Examples:**

**examples/main\_llist.c.**

**4.3.4.5 `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 `_gdsl_list_t`.

**Parameters**

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

**Examples:**

`examples/main_lllist.c`.

#### 4.3.4.6 `void _gdsl_list_insert_after( _gdsl_list_t L, _gdsl_list_t PREV )`

Insert a low-level list after another one.

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

**Note**

Complexity:  $O(|L|)$

**Precondition**

L & PREV must be non-empty `_gdsl_list_t`.

**Parameters**

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

**See also**

`_gdsl_list_insert_before()` (p. 52)

`_gdsl_list_remove()` (p. 53)

#### 4.3.4.7 `void _gdsl_list_insert_before( _gdsl_list_t L, _gdsl_list_t SUCC )`

Insert a low-level list before another one.

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

**Note**

Complexity:  $O(|L|)$

**Precondition**

L & SUCC must be non-empty `_gdsl_list_t`.

**Parameters**

<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**

`_gdsl_list_insert_after()` (p. 52)

`_gdsl_list_remove()` (p. 53)

**4.3.4.8 void `_gdsl_list_remove( _gdsl_node_t NODE )`**

Remove a node from a low-level list.

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

**Note**

Complexity:  $O(1)$

**Precondition**

`NODE` must be a non-empty `_gdsl_node_t`.

**Parameters**

<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. 52)

`_gdsl_list_insert_before()` (p. 52)

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

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

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

**Note**

Complexity:  $O(|L|)$

**Precondition**

COMP\_F != NULL

**Parameters**

<i>L</i>	The low-level list to use
<i>COMP_F</i>	The comparison function to use to compare L'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 sub-list starting by e if it's found.  
 NULL if VALUE is not found in L.

#### 4.3.4.10 `_gdsl_list_t _gdsl_list_map_forward( const _gdsl_list_t L, const _gdsl_node_map_func_t MAP_F, void * USER_DATA )`

Parse a low-level list in forward order.

Parse all nodes of the low-level list L in forward order. The MAP\_F function is called on each node with the USER\_DATA argument. If MAP\_F returns GDSL\_MAP\_STOP, then `_gdsl_list_map_forward()` (p. 54) stops and returns its last 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 GDSL\_MAP\_STOP.  
 NULL when the parsing is done.



See also

`_gdsl_list_map_backward()` (p. 55)

Examples:

`examples/main_llist.c`.

4.3.4.11 `_gdsl_list_t _gdsl_list_map_backward( const _gdsl_list_t L, const  
_gdsl_node_map_func_t MAP_F, void * USER_DATA )`

Parse a low-level list in backward order.

Parse all nodes of the low-level list L in backward order. The MAP\_F function is called on each node with the USER\_DATA argument. If MAP\_F returns GDSDL\_MAP\_STOP, then `_gdsl_list_map_backward()` (p. 55) 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

`_gdsl_list_map_forward()` (p. 54)

Examples:

`examples/main_llist.c`.

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

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

Write the nodes of the low-level list *L* to *OUTPUT\_FILE*, using *WRITE\_F* function. -  
 Additionnal *USER\_DATA* argument could be passed to *WRITE\_F*.

#### Note

Complexity:  $O(|L|)$

#### Precondition

*WRITE\_F* != NULL & *OUTPUT\_FILE* != NULL.

#### Parameters

<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 <i>L</i> 's nodes.
<i>USER_DATA</i>	User's datas passed to <i>WRITE_F</i> .

#### See also

[\\_gdsI\\_list\\_write\\_xml\(\)](#) (p. 56)

[\\_gdsI\\_list\\_dump\(\)](#) (p. 57)

#### Examples:

**examples/main\_Illist.c.**

```
4.3.4.13 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 WRITE_F.

See also

`_gdsi_list_write()` (p. 55)  
`_gdsi_list_dump()` (p. 57)

Examples:

`examples/main_llist.c`.

4.3.4.14 `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 WRITE_F.

See also

`_gdsi_list_write()` (p. 55)

`_gdsi_list_write_xml()` (p. 56)

Examples:

`examples/main_llist.c.`

## 4.4 Low-level doubly-linked node manipulation module

### 4.4.1

### 4.4.2 Copyright

Copyright (C) 1998-2017 Nicolas Darnis <ndarnis@free.fr>.

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

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

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

### Typedefs

- typedef struct \_gdsl\_node \* **\_gdsl\_node\_t**  
*GDSL low-level doubly linked node type.*
- typedef int(\* **\_gdsl\_node\_map\_func\_t**)(const **\_gdsl\_node\_t** NODE, void \*USER\_DATA)  
*GDSL low-level doubly-linked node map function type.*
- typedef void(\* **\_gdsl\_node\_write\_func\_t**)(const **\_gdsl\_node\_t** NODE, FILE \*OUTPUT\_FILE, void \*USER\_DATA)  
*GDSL low-level doubly-linked node write function type.*

### Functions

- **\_gdsl\_node\_t \_gdsl\_node\_alloc**(void)  
*Create a new low-level node.*
- **gdsl\_element\_t \_gdsl\_node\_free**(**\_gdsl\_node\_t** NODE)  
*Destroy a low-level node.*
- **\_gdsl\_node\_t \_gdsl\_node\_get\_succ**(const **\_gdsl\_node\_t** NODE)  
*Get the successor of a low-level node.*
- **\_gdsl\_node\_t \_gdsl\_node\_get\_pred**(const **\_gdsl\_node\_t** NODE)  
*Get the predecessor of a low-level node.*
- **gdsl\_element\_t \_gdsl\_node\_get\_content**(const **\_gdsl\_node\_t** NODE)  
*Get the content of a low-level node.*
- void **\_gdsl\_node\_set\_succ**(**\_gdsl\_node\_t** NODE, const **\_gdsl\_node\_t** SUC-C)  
*Set the successor of a low-level node.*

- void **\_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.*

### 4.4.3 Typedef Documentation

#### 4.4.3.1 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 69 of file **\_gdsI\_node.h**.

#### 4.4.3.2 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

<i>NODE</i>	The low-level node 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 78 of file **\_gdsI\_node.h**.

4.4.3.3 `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

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

Definition at line 88 of file `_gdsI_node.h`.

### 4.4.4 Function Documentation

#### 4.4.4.1 `_gdsI_node_t _gdsI_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

`_gdsI_node_free()` (p. 61)

#### 4.4.4.2 `gdsI_element_t _gdsI_node_free( _gdsI_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**

`_gdsi_node_alloc()` (p. 61)

#### 4.4.4.3 `_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. 62)

`_gdsi_node_set_succ()` (p. 64)

`_gdsi_node_set_pred()` (p. 64)

#### 4.4.4.4 `_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**

[`\_gdsi\_node\_get\_succ\(\)`](#) (p. 62)

[`\_gdsi\_node\_set\_succ\(\)`](#) (p. 64)

[`\_gdsi\_node\_set\_pred\(\)`](#) (p. 64)

**4.4.4.5 `_gdsi_element_t _gdsi_node_get_content( const _gdsi_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

`_gdsi_node_set_content()` (p. 65)

Examples:

`examples/main_llist.c`.

4.4.4.6 `void _gdsi_node_set_succ( _gdsi_node_t NODE, const _gdsi_node_t SUCC )`

Set the successor of a low-level node.

Modify 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. 62)

4.4.4.7 `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. 62)

4.4.4.8 `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. 63)

4.4.4.9 `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

`_gdsl_node_unlink()` (p. 66)

4.4.4.10 `void _gdsl_node_unlink( _gdsl_node_t NODE1, _gdsl_node_t NODE2 )`

Unlink two low-level nodes.

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

Note

Complexity:  $O(1)$

Precondition

`NODE1 != NULL & NODE2 != NULL`

Parameters

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

See also

`_gdsl_node_link()` (p. 65)

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

Write a low-level node to a file.

Write the low-level node `NODE` to `OUTPUT_FILE`, using `WRITE_F` function. Additional `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. 67)

[`\_gdsi\_node\_dump\(\)`](#) (p. 68)

4.4.4.12 `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. 66)

[`\_gdsi\_node\_dump\(\)`](#) (p. 68)

4.4.4.13 void `_gdsl_node_dump` ( const `_gdsl_node_t` *NODE*, const  
`_gdsl_node_write_func_t` *WRITE\_F*, FILE \* *OUTPUT\_FILE*, void \* *USER\_DATA* )

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

Dump the structure of the low-level node *NODE* to *OUTPUT\_FILE*. If *WRITE\_F* != NULL, then uses *WRITE\_F* function to write *NODE* to *OUTPUT\_FILE*. Additionnal *USER\_DATA* argument could be passed to *WRITE\_F*.

#### Note

Complexity:  $O(1)$

#### Precondition

*NODE* != NULL & *OUTPUT\_FILE* != NULL

#### Parameters

<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

`_gdsl_node_write()` (p. 66)

`_gdsl_node_write_xml()` (p. 67)

## 4.5 Main module

### 4.5.1

### 4.5.2 Copyright

Copyright (C) 1998-2017 Nicolas Darnis <ndarnis@free.fr>.

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

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

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

### Functions

- const char \* **gdsl\_get\_version** (void)  
*Get GDSDL version number as a string.*

### 4.5.3 Function Documentation

#### 4.5.3.1 const char\* gdsl\_get\_version ( void )

Get GDSDL version number as a string.

#### Note

Complexity: O( 1 )

#### Precondition

nothing.

#### Postcondition

the returned string MUST NOT be deallocated.

#### Returns

the GDSDL version number as a string.

## 4.6 2D-Arrays manipulation module

### 4.6.1

### 4.6.2 Copyright

Copyright (C) 1998-2017 Nicolas Darnis <ndarnis@free.fr>.

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

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

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

### 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.*

### 4.6.3 Typedef Documentation

#### 4.6.3.1 typedef struct gdsI\_2darray\* gdsI\_2darray\_t

GDSL 2D-array type.

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

Definition at line 69 of file gdsI\_2darray.h.

### 4.6.4 Function Documentation

#### 4.6.4.1 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.

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. 72)  
**gdsI\_alloc\_func\_t** (p. 243)  
**gdsI\_free\_func\_t** (p. 243)

## 4.6.4.2 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. 71) 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

<i>A</i>	The 2D-array to destroy
----------	-------------------------

## See also

**gdsI\_2darray\_alloc()** (p. 71)

## 4.6.4.3 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. 75)

**4.6.4.4 `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. 74)

**gdsI\_2darray\_get\_size()** (p. 74)

#### 4.6.4.5 `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 `gdsI_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

**gdsI\_2darray\_get\_rows\_number()** (p. 73)

**gdsI\_2darray\_get\_size()** (p. 74)

#### 4.6.4.6 `ulong gdsI_2darray_get_size ( const gdsI_2darray_t A )`

Get the size of a 2D-array.

Note

Complexity:  $O(1)$

Precondition

A must be a valid `gdsI_2darray_t`

Parameters

A	The 2D-array to use.
---	----------------------

**Returns**

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

**See also**

**gdsI\_2darray\_get\_rows\_number()** (p. 73)

**gdsI\_2darray\_get\_columns\_number()** (p. 74)

**4.6.4.7 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.

**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. 76)

**4.6.4.8 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**

**`gdsi_2darray_get_name()`** (p. 72)

#### 4.6.4.9 `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.

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 `gdsi_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 `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 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**

**gdsI\_2darray\_get\_content()** (p. 75)

**4.6.4.10** 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.

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. 77)

**gdsI\_2darray\_dump()** (p. 78)

**4.6.4.11** 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 `gdsl_2darray_t` & `OUTPUT_FILE` != NULL

**Parameters**

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

**See also**

**`gdsl_2darray_write()`** (p. 77)

**`gdsl_2darray_dump()`** (p. 78)

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

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

Dump  $A$ 's structure to `OUTPUT_FILE`. If `WRITE_F` != NULL, then uses `WRITE_F` to write  $A$ 's elements to `OUTPUT_FILE`. Additionnal `USER_DATA` argument could be passed to `WRITE_F`.

**Note**

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

**Precondition**

$A$  must be a valid `gdsl_2darray_t` & `OUTPUT_FILE` != NULL

**Parameters**

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



See also

**`gdsi_2darray_write()`** (p. 77)

**`gdsi_2darray_write_xml()`** (p. 77)

## 4.7 Binary search tree manipulation module

### 4.7.1

### 4.7.2 Copyright

Copyright (C) 1998-2017 Nicolas Darnis <ndarnis@free.fr>.

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

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

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

### 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.*

- **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.*
- **gdsi\_element\_t gdsi\_bstree\_remove** (**gdsi\_bstree\_t** T, void \*VALUE)  
*Remove an element from a binary search tree.*
- **gdsi\_bstree\_t gdsi\_bstree\_delete** (**gdsi\_bstree\_t** T, void \*VALUE)  
*Delete an element from a binary search tree.*
- **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.*
- **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.*
- **gdsi\_element\_t gdsi\_bstree\_map\_infix** (const **gdsi\_bstree\_t** T, **gdsi\_map\_func\_t** MAP\_F, void \*USER\_DATA)  
*Parse a binary search tree in infix order.*
- **gdsi\_element\_t gdsi\_bstree\_map\_postfix** (const **gdsi\_bstree\_t** T, **gdsi\_map\_func\_t** MAP\_F, void \*USER\_DATA)  
*Parse a binary search tree in postfix order.*
- void **gdsi\_bstree\_write** (const **gdsi\_bstree\_t** T, **gdsi\_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 **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.*
- 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.*

### 4.7.3 Typedef Documentation

#### 4.7.3.1 typedef struct gdsi\_bstree\* gdsi\_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 72 of file gdsi\_bstree.h.

### 4.7.4 Function Documentation

#### 4.7.4.1 gdsi\_bstree\_t gdsi\_bstree\_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 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. 82)  
`gdsI_bstree_flush()` (p. 83)  
`gdsI_alloc_func_t` (p. 243)  
`gdsI_free_func_t` (p. 243)  
`gdsI_compare_func_t` (p. 245)

#### Examples:

`examples/main_bstree.c`.

#### 4.7.4.2 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. 81). The name of `T` is deallocated and `T` is deallocated itself too.

**Note**

Complexity:  $O(|T|)$

**Precondition**

T must be a valid `gdsl_bstree_t`

**Parameters**

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

**See also**

**`gdsl_bstree_alloc()`** (p. 81)

**`gdsl_bstree_flush()`** (p. 83)

**Examples:**

**`examples/main_bstree.c`.**

**4.7.4.3 void `gdsl_bstree_flush ( gsdl_bstree_t T )`**

Flush a binary search tree.

Deallocate all the elements of the binary search tree T by calling T's `FREE_F` function passed to **`gdsl_rbtree_alloc()`** (p. 210). The binary search tree T is not deallocated itself and its name is not modified.

**Note**

Complexity:  $O(|T|)$

**Precondition**

T must be a valid `gdsl_bstree_t`

**Parameters**

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

**See also**

**`gdsl_bstree_alloc()`** (p. 81)

**`gdsl_bstree_free()`** (p. 82)

**Examples:**

**`examples/main_bstree.c`.**

#### 4.7.4.4 `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. 86) ()

#### 4.7.4.5 `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

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

**Returns**

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

**Examples:**

**examples/main\_bstree.c.**

**4.7.4.6 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**

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

**Returns**

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

**Examples:**

**examples/main\_bstree.c.**

**4.7.4.7 ulong 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**

<i>T</i>	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. 86)

**Examples:**

**examples/main\_bstree.c.**

#### 4.7.4.8 `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**

<i>T</i>	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. 85)

**Examples:**

**examples/main\_bstree.c.**

#### 4.7.4.9 `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

<i>T</i>	The binary search tree to change the name
<i>NEW_NAME</i>	The new name of <i>T</i>

#### Returns

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

#### See also

**`gdsi_bstree_get_name()`** (p. 84)

#### 4.7.4.10 `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 <i>T</i>
<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\_bstree\_remove()** (p. 88)

**gdsl\_bstree\_delete()** (p. 89)

**Examples:**

**examples/main\_bstree.c.**

**4.7.4.11 gsdl\_element\_t gsdl\_bstree\_remove( gsdl\_bstree\_t T, void \* VALUE )**

Remove an element from a binary search tree.

Remove from the binary search tree T the first founded element E equal to VALUE, by using T's COMP\_F function passed to **gdsl\_bstree\_alloc()** (p. 81). 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 gsdl\_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**

**gdsl\_bstree\_insert()** (p. 87)

**gdsl\_bstree\_delete()** (p. 89)

4.7.4.12 `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. 81). 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. 81), 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**

<i>T</i>	The binary search tree to remove an element from
<i>VALUE</i>	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. 87)  
**gdsi\_bstree\_remove()** (p. 88)

**Examples:**

**examples/main\_bstree.c.**

4.7.4.13 `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. 81) 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. 87)  
**`gdsi_bstree_remove()`** (p. 88)  
**`gdsi_bstree_delete()`** (p. 89)

**Examples:**

**`examples/main_bstree.c`.**

#### 4.7.4.14 `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. 90) 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>	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. 91)  
**gdsl\_bstree\_map\_postfix()** (p. 92)

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

Parse a binary search tree in 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. 91) 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. 90)  
**gdsl\_bstree\_map\_postfix()** (p. 92)

#### 4.7.4.16 `gdsl_element_t gsdl_bstree_map_postfix( const gsdl_bstree_t T, gsdl_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 **gsdl\_bstree\_map\_postfix()** (p. 92) stops and returns its last examined element.

##### Note

Complexity:  $O(|T|)$

##### Precondition

T must be a valid `gsdl_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

**gsdl\_bstree\_map\_prefix()** (p. 90)  
**gsdl\_bstree\_map\_infix()** (p. 91)

#### 4.7.4.17 `void gsdl_bstree_write( const gsdl_bstree_t T, gsdl_write_func_t WRITE_F, FILE * OUTPUT_FILE, void * USER_DATA )`

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

Write the nodes elements of the binary search tree T to OUTPUT\_FILE, using WRITE\_F function. Additionnal USER\_DATA argument could be passed to WRITE\_F.

##### Note

Complexity:  $O(|T|)$

**Precondition**

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

**Parameters**

<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**

**`gdsl_bstree_write_xml()`** (p. 93)

**`gdsl_bstree_dump()`** (p. 94)

**Examples:**

**`examples/main_bstree.c`.**

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

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

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

**Note**

Complexity:  $O(|T|)$

**Precondition**

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

**Parameters**

<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. 92)  
**gdsi\_bstree\_dump()** (p. 94)

Examples:

**examples/main\_bstree.c.**

**4.7.4.19** 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 <i>T</i> 's elements.
<i>USER_DATA</i>	User's datas passed to <i>WRITE_F</i> .

See also

**gdsi\_bstree\_write()** (p. 92)  
**gdsi\_bstree\_write\_xml()** (p. 93)

Examples:

**examples/main\_bstree.c.**



## 4.8 Hashtable manipulation module

### 4.8.1

### 4.8.2 Copyright

Copyright (C) 1998-2017 Nicolas Darnis <ndarnis@free.fr>.

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

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

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

### 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.*
- **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.*

### 4.8.3 Typedef Documentation

#### 4.8.3.1 typedef struct hash\_table\* gdsi\_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 72 of file gdsi\_hash.h.

#### 4.8.3.2 typedef const char\*(\* gdsI\_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 80 of file gdsI\_hash.h.

#### 4.8.3.3 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 88 of file gdsI\_hash.h.

### 4.8.4 Function Documentation

#### 4.8.4.1 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.

4.8.4.2 **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. 107) 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. 97) 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. 99)  
**gdsI\_hash\_flush()** (p. 99)  
**gdsI\_hash\_insert()** (p. 104)  
**gdsI\_hash\_modify()** (p. 107)

**Examples:**

**examples/main\_hash.c.**

**4.8.4.3 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. 98). 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. 98)  
**gdsI\_hash\_flush()** (p. 99)

**Examples:**

**examples/main\_hash.c.**

**4.8.4.4 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. 98). *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. 98)

**`gdsI_hash_free()`** (p. 99)

**Examples:**

**`examples/main_hash.c`**.

#### 4.8.4.5 `const char* gdsI_hash_get_name( const gdsI_hash_t H )`

Get the name of a hashtable.

**Note**

Complexity:  $O(1)$

**Precondition**

*H* must be a valid `gdsI_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**

**gdsI\_hash\_set\_name()** (p. 104)

**4.8.4.6 ushort gdsI\_hash\_get\_entries\_number( const gdsI\_hash\_t H )**

Get the number of entries 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**

the number of lists entries of the hashtable H.

**See also**

**gdsI\_hash\_get\_size()** (p. 102)  
**gdsI\_hash\_fill\_factor()**

**4.8.4.7 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 **gdsl\_hash\_modify()** (p. 107) was used with a NEW\_LISTS\_MAX\_SIZE greather than the actual one.

**See also**

**gdsl\_hash\_fill\_factor()**  
**gdsl\_hash\_get\_entries\_number()** (p. 101)  
**gdsl\_hash\_get\_longest\_list\_size()** (p. 102)  
**gdsl\_hash\_modify()** (p. 107)

**4.8.4.8 ushort gdsl\_hash\_get\_longest\_list\_size( const gdsl\_hash\_t H )**

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

**Note**

Complexity:  $O(L)$ , where  $L = \text{gdsl\_hash\_get\_entries\_number}(H)$

**Precondition**

H must be a valid gdsl\_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**

**gdsl\_hash\_get\_size()** (p. 102)  
**gdsl\_hash\_fill\_factor()**  
**gdsl\_hash\_get\_entries\_number()** (p. 101)  
**gdsl\_hash\_get\_lists\_max\_size()** (p. 101)

**4.8.4.9 ulong gdsl\_hash\_get\_size( const gdsl\_hash\_t H )**

Get the size of a hashtable.



**Note**

Complexity:  $O(L)$ , where  $L = \text{gdsI\_hash\_get\_entries\_number}(H)$

**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. 101)

`gdsI_hash_fill_factor()`

**`gdsI_hash_get_longest_list_size()`** (p. 102)

**4.8.4.10 `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. 101)

**`gdsI_hash_get_longest_list_size()`** (p. 102)

**`gdsI_hash_get_size()`** (p. 102)

Examples:

**examples/main\_hash.c.**

**4.8.4.11** `gdsl_hash_t gsdl_hash_set_name( gsdl_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 `gsdl_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

**`gsdl_hash_get_name()`** (p. 100)

**4.8.4.12** `gsdl_element_t gsdl_hash_insert( gsdl_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 `gsdl_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 `gsdl_hash_get_entries_number(H)` (say N) is modified as  $N * 2 + 1$
- its actual `gsdl_hash_get_lists_max_size(H)` (say M) is modified as  $M * 2$  The element E is then inserted into H at the entry computed by `HASH_F( K ) modulo gsdl_hash_get_entries_number(H)`. `ALLOC_F`, `KEY_F` and `HASH_F` are the function pointers passed to **`gsdl_hash_alloc()`** (p. 98).

**Note**

Complexity:  $O(1)$  if `gdsI_hash_get_lists_max_size(H)` is not reached or if it is equal to zero

Complexity:  $O(\text{gdsI_hash_modify}(H))$  if `gdsI_hash_get_lists_max_size(H)` is reached, so `H` needs to grow

**Precondition**

`H` must be a valid `gdsI_hash_t`

**Parameters**

<i>H</i>	The hashtable to modify
<i>VALUE</i>	The value used to make the new element to insert into <code>H</code>

**Returns**

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

**See also**

**`gdsI_hash_alloc()`** (p. 98)  
**`gdsI_hash_remove()`** (p. 105)  
**`gdsI_hash_delete()`** (p. 106)  
**`gdsI_hash_get_size()`** (p. 102)  
**`gdsI_hash_get_entries_number()`** (p. 101)  
**`gdsI_hash_modify()`** (p. 107)

**Examples:**

**`examples/main_hash.c`.**

**4.8.4.13 `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. 104)  
**gdsI\_hash\_search()** (p. 108)  
**gdsI\_hash\_delete()** (p. 106)

## Examples:

**examples/main\_hash.c.**

**4.8.4.14 gdsI\_hash\_t gdsI\_hash\_delete( gdsI\_hash\_t *H*, const char \* *KEY* )**

Delete an element from a hashtable.

Remove from the 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. 98), then *H* is 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 modified hashtable after removal of *E* if *E* was found.  
NULL if no element equal to *KEY* was found.

See also

**gdsI\_hash\_insert()** (p. 104)

**gdsI\_hash\_search()** (p. 108)

**gdsI\_hash\_remove()** (p. 105)

**4.8.4.15 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.

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 **gdsI\_hash\_insert()** (p. 104).

Note

Complexity:  $O(|H|)$

Precondition

H must be a valid gdsI\_hash\_t & NEW\_ENTRIES\_NB > gdsI\_hash\_get\_entries\_number(H) & NEW\_LISTS\_MAX\_SIZE > gdsI\_hash\_get\_lists\_max\_size(H)

Parameters

<i>H</i>	The hashtable to modify
<i>NEW_ENTRIES_NB</i>	
<i>NEW_LISTS_MAX_SIZE</i>	

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. 104)

**gdsI\_hash\_get\_entries\_number()** (p. 101)

**gdsI\_hash\_get\_fill\_factor()** (p. 103)

**gdsI\_hash\_get\_longest\_list\_size()** (p. 102)

**gdsI\_hash\_get\_lists\_max\_size()** (p. 101)

#### 4.8.4.16 `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. 104)

**`gdsi_hash_remove()`** (p. 105)

**`gdsi_hash_delete()`** (p. 106)

##### Examples:

**`examples/main_hash.c`**.

#### 4.8.4.17 `gdsi_element_t gdsi_hash_map( const gdsi_hash_t H, gdsi_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 **`gdsi_hash_map()`** (p. 108) 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 <code>MAP_F</code>

**Returns**

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

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

Write all the elements of a hashtable to a file.

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

**Note**

Complexity:  $O(|H|)$

**Precondition**

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

**Parameters**

<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 <code>WRITE_F</code> .

**See also**

**`gdsl_hash_write_xml()`** (p. 110)

**`gdsl_hash_dump()`** (p. 110)

**Examples:**

**`examples/main_hash.c`.**

4.8.4.19 `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. Additional 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. 109)  
**gdsi\_hash\_dump()** (p. 110)

#### Examples:

**examples/main\_hash.c.**

4.8.4.20 `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. Additional 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 <code>WRITE_F</code>

## See also

**`gdsi_hash_write()`** (p. 109)

**`gdsi_hash_write_xml()`** (p. 110)

## Examples:

**`examples/main_hash.c`**.

## 4.9 Heap manipulation module

### 4.9.1

### 4.9.2 Copyright

Copyright (C) 1998-2017 Nicolas Darnis <ndarnis@free.fr>.

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

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

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

### Typedefs

- typedef struct heap \* **gdsl\_heap\_t**

*GDSL heap type.*

### Functions

- **gdsl\_heap\_t gsdl\_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 gsdl\_heap\_get\_size** (const **gdsl\_heap\_t** H)

*Get the size of a heap.*

- **gdsl\_element\_t gsdl\_heap\_get\_top** (const **gdsl\_heap\_t** H)

*Get the top of a heap.*

- **bool gsdl\_heap\_is\_empty** (const **gdsl\_heap\_t** H)

*Check if a heap is empty.*

- **gdsl\_heap\_t gsdl\_heap\_set\_name** (**gdsl\_heap\_t** H, const char \*NEW\_NAME)

*Set the name of a heap.*

- **gdsl\_element\_t gsdl\_heap\_set\_top** (**gdsl\_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.*

### 4.9.3 Typedef Documentation

#### 4.9.3.1 typedef struct heap\* gdsi\_heap\_t

GDSL heap 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 73 of file gdsi\_heap.h.

### 4.9.4 Function Documentation

#### 4.9.4.1 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**

<i>NAME</i>	The name of the new heap to create
<i>ALLOC_F</i>	Function to alloc element when inserting it in the heap
<i>FREE_F</i>	Function to free element when removing it from the heap
<i>COMP_F</i>	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. 114)  
**gdsI\_heap\_flush()** (p. 115)

**Examples:**

**examples/main\_heap.c.**

**4.9.4.2 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. 113). 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. 113)

**gdsI\_heap\_flush()** (p. 115)

Examples:

**examples/main\_heap.c.**

#### 4.9.4.3 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. 113). *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. 113)

**gdsI\_heap\_free()** (p. 114)

Examples:

**examples/main\_heap.c.**

#### 4.9.4.4 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. 118)

**Examples:**

**examples/main\_heap.c.**

**4.9.4.5   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 gdsI\_heap\_t

**Parameters**

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

**Returns**

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

**4.9.4.6   gdsI\_element\_t gdsI\_heap\_get\_top( const gdsI\_heap\_t *H* )**

Get the top of a heap.

**Note**

Complexity:  $O(1)$

**Precondition**

H must be a valid `gdsI_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**

**`gdsI_heap_set_top()`** (p. 118)

**Examples:**

**`examples/main_heap.c`.**

**4.9.4.7 `bool gdsI_heap_is_empty( const gdsI_heap_t H )`**

Check if a heap is empty.

**Note**

Complexity:  $O(1)$

**Precondition**

H must be a valid `gdsI_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.

**Examples:**

**examples/main\_heap.c.**

**4.9.4.8 gdsI\_heap\_t gdsI\_heap\_set\_name( gdsI\_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 gdsI\_heap\_t

**Parameters**

<i>H</i>	The heap to change the name
<i>NEW_NAME</i>	The new name of <i>H</i>

**Returns**

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

**See also**

**gdsI\_heap\_get\_name()** (p. 115)

**4.9.4.9 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. 116)

**Examples:**

**`examples/main_heap.c`.**

**4.9.4.10 `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. 113)  
**gdsI\_heap\_remove()**  
**gdsI\_heap\_delete()**  
**gdsI\_heap\_get\_size()** (p. 116)

Examples:

**examples/main\_heap.c.**

#### 4.9.4.11 **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. 119)  
**gdsI\_heap\_delete\_top()** (p. 120)

#### 4.9.4.12 **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. 113), 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. 119)  
**`gdsI_heap_remove_top()`** (p. 120)

**Examples:**

**`examples/main_heap.c`.**

**4.9.4.13** `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.

**Examples:**

**examples/main\_heap.c.**

**4.9.4.14** 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.

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 gdsI\_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 H's elements.
<i>USER_DATA</i>	User's datas passed to WRITE_F.

**See also**

**gdsI\_heap\_write\_xml()** (p. 122)

**gdsI\_heap\_dump()** (p. 123)

**4.9.4.15** 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.

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. 122)

**`gdsI_heap_dump()`** (p. 123)

**Examples:**

**`examples/main_heap.c`.**

**4.9.4.16** `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. 122)

**gdsI\_heap\_write\_xml()** (p. 122)

Examples:

**examples/main\_heap.c.**

## 4.10 Interval Heap manipulation module

### 4.10.1

### 4.10.2 Copyright

Copyright (C) 1998-2017 Nicolas Darnis <ndarnis@free.fr>.

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

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

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

### Typedefs

- typedef struct heap \* **gdsl\_interval\_heap\_t**

*GDSDL 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.*

### 4.10.3 Typedef Documentation

#### 4.10.3.1 typedef struct heap\* **gdsl\_interval\_heap\_t**

GDSL interval heap type.

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

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



#### 4.10.4 Function Documentation

4.10.4.1 `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.

Allocate a new interval heap data structure which name is set to a copy of NAME. The function pointers ALLOC\_F, FREE\_F and COMP\_F could be used to respectively, alloc, free and compares elements in the interval heap. These pointers could be set to NULL to use the default ones:

- the default ALLOC\_F simply returns its argument
- the default FREE\_F does nothing
- the default COMP\_F always returns 0

##### Note

Complexity:  $O(1)$

##### Precondition

nothing

##### Parameters

NAME	The name of the new interval heap to create
ALLOC_F	Function to alloc element when inserting it in the interval heap
FREE_F	Function to free element when removing it from the interval heap
COMP_F	Function to compare elements into the interval heap

##### Returns

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

##### See also

`gdsI_interval_heap_free()` (p. 128)  
`gdsI_interval_heap_flush()` (p. 128)

##### Examples:

`examples/main_interval_heap.c`.

#### 4.10.4.2 void `gdsl_interval_heap_free`( `gdsl_interval_heap_t` *H* )

Destroy an interval heap.

Deallocate all the elements of the interval heap *H* by calling *H*'s `FREE_F` function passed to **`gdsl_interval_heap_alloc`**( p. 127). The name of *H* is deallocated and - *H* is deallocated itself too.

##### Note

Complexity:  $O(|H|)$

##### Precondition

*H* must be a valid `gdsl_interval_heap_t`

##### Parameters

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

##### See also

**`gdsl_interval_heap_alloc`**( p. 127)

**`gdsl_interval_heap_flush`**( p. 128)

##### Examples:

**`examples/main_interval_heap.c`**.

#### 4.10.4.3 void `gdsl_interval_heap_flush`( `gdsl_interval_heap_t` *H* )

Flush an interval heap.

Deallocate all the elements of the interval heap *H* by calling *H*'s `FREE_F` function passed to **`gdsl_interval_heap_alloc`**( p. 127). *H* is not deallocated itself and *H*'s name is not modified.

##### Note

Complexity:  $O(|H|)$

##### Precondition

*H* must be a valid `gdsl_interval_heap_t`

##### Parameters

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

See also

**gdsI\_interval\_heap\_alloc()** (p. 127)

**gdsI\_interval\_heap\_free()** (p. 128)

Examples:

**examples/main\_interval\_heap.c.**

#### 4.10.4.4 `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

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

Returns

the name of the interval heap H.

See also

**gdsI\_interval\_heap\_set\_name()** (p. 131)

#### 4.10.4.5 `ulong gdsI_interval_heap_get_size( const gdsI_interval_heap_t H )`

Get the size of a interval heap.

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.

**4.10.4.8 `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</i>	The new name of H

**Returns**

the modified interval heap in case of success.

NULL in case of insufficient memory.

**See also**

**`gdsI_interval_heap_get_name()`** (p. 129)

**4.10.4.9 `gdsI_element_t gdsI_interval_heap_insert ( gdsI_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. 127)  
`gdsl_interval_heap_remove()`  
`gdsl_interval_heap_delete()`  
**`gdsl_interval_heap_get_size()`** (p. 129)

#### Examples:

**`examples/main_interval_heap.c.`**

#### 4.10.4.10 `gdsl_element_t` `gdsl_interval_heap_remove_max` ( `gdsl_interval_heap_t H` )

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

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

#### Note

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

#### Precondition

H must be a valid `gdsl_interval_heap_t`

## Parameters

<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. 131)  
**gdsI\_interval\_heap\_delete\_max()** (p. 135)

## Examples:

**examples/main\_interval\_heap.c.**

#### 4.10.4.11 **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. 131)  
**gdsI\_interval\_heap\_delete\_max()** (p. 135)

Examples:

`examples/main_interval_heap.c.`

**4.10.4.12** `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

**4.10.4.13** `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



#### 4.10.4.14 `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. 127), 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. 131)  
`gdsI_interval_heap_remove_top()`

#### 4.10.4.15 `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. 127), 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

**gdsl\_interval\_heap\_insert()** (p. 131)  
**gdsl\_interval\_heap\_remove\_top()**

**4.10.4.16 gdsl\_element\_t gdsl\_interval\_heap\_map\_forward ( const  
 gdsl\_interval\_heap\_t *H*, gdsl\_map\_func\_t *MAP\_F*, void \* *USER\_DATA* )**

Parse a interval heap.

Parse all elements of the interval heap *H*. The *MAP\_F* function is called on each *H*'s element with *USER\_DATA* argument. If *MAP\_F* returns **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 <i>MAP_F</i>

## Returns

the first element for which *MAP\_F* returns **GD\_SL\_MAP\_STOP**.  
 NULL when the parsing is done.

**4.10.4.17 void gdsl\_interval\_heap\_write ( const gdsl\_interval\_heap\_t *H*,  
 gdsl\_write\_func\_t *WRITE\_F*, FILE \* *OUTPUT\_FILE*, void \* *USER\_DATA* )**

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

Write the elements of the interval heap *H* to *OUTPUT\_FILE*, using *WRITE\_F* function. Additionnal *USER\_DATA* argument could be passed to *WRITE\_F*.

**Note**

Complexity:  $O(|H|)$

**Precondition**

*H* must be a valid *gdsi\_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**

***gdsi\_interval\_heap\_write\_xml()*** (p. 137)

***gdsi\_interval\_heap\_dump()*** (p. 138)

**4.10.4.18** void *gdsi\_interval\_heap\_write\_xml*( const *gdsi\_interval\_heap\_t* *H*,  
*gdsi\_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 <i>H</i> 's elements.
<i>USER_DATA</i>	User's datas passed to <i>WRITE_F</i> .

See also

**gdsI\_interval\_heap\_write()** (p. 136)

**gdsI\_interval\_heap\_dump()** (p. 138)

4.10.4.19 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 <i>H</i> 's elements
<i>USER_DATA</i>	User's datas passed to <i>WRITE_F</i>

See also

**gdsI\_interval\_heap\_write()** (p. 136)

**gdsI\_interval\_heap\_write\_xml()** (p. 137)

## 4.11 Doubly-linked list manipulation module

### 4.11.1

#### 4.11.2 Copyright

Copyright (C) 1998-2017 Nicolas Darnis <ndarnis@free.fr>.

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

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

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

#### 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.*

- **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.*

- **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.*

- **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.*

- **void gdsi\_list\_write** (const gdsi\_list\_t L, gdsi\_write\_func\_t WRITE\_F, FILE \*OUTPUT\_FILE, void \*USER\_DATA)

*Write all the elements of a list to a file.*

- **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.*

- void **gds\_l\_dump** (const **gds\_l\_t** L, **gds\_l\_write\_func\_t** WRITE\_F, FILE \*OUTPUT\_FILE, void \*USER\_DATA)

*Dump the internal structure of a list to a file.*

- **gds\_l\_cursor\_t** **gds\_l\_cursor\_alloc** (const **gds\_l\_t** L)

*Create a new list cursor.*

- void **gds\_l\_cursor\_free** (**gds\_l\_cursor\_t** C)

*Destroy a list cursor.*

- void **gds\_l\_cursor\_move\_to\_head** (**gds\_l\_cursor\_t** C)

*Put a cursor on the head of its list.*

- void **gds\_l\_cursor\_move\_to\_tail** (**gds\_l\_cursor\_t** C)

*Put a cursor on the tail of its list.*

- **gds\_l\_element\_t** **gds\_l\_cursor\_move\_to\_value** (**gds\_l\_cursor\_t** C, **gds\_l\_compare\_func\_t** COMP\_F, void \*VALUE)

*Place a cursor on a particular element.*

- **gds\_l\_element\_t** **gds\_l\_cursor\_move\_to\_position** (**gds\_l\_cursor\_t** C, **ulong** POS)

*Place a cursor on a element given by its position.*

- void **gds\_l\_cursor\_step\_forward** (**gds\_l\_cursor\_t** C)

*Move a cursor one step forward of its list.*

- void **gds\_l\_cursor\_step\_backward** (**gds\_l\_cursor\_t** C)

*Move a cursor one step backward of its list.*

- **bool** **gds\_l\_cursor\_is\_on\_head** (const **gds\_l\_cursor\_t** C)

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

- **bool** **gds\_l\_cursor\_is\_on\_tail** (const **gds\_l\_cursor\_t** C)

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

- **bool** **gds\_l\_cursor\_has\_succ** (const **gds\_l\_cursor\_t** C)

*Check if a cursor has a successor.*

- **bool** **gds\_l\_cursor\_has\_pred** (const **gds\_l\_cursor\_t** C)

*Check if a cursor has a predecessor.*

- void **gds\_l\_cursor\_set\_content** (**gds\_l\_cursor\_t** C, **gds\_l\_element\_t** E)

*Set the content of the cursor.*

- **gds\_l\_element\_t** **gds\_l\_cursor\_get\_content** (const **gds\_l\_cursor\_t** C)

*Get the content of a cursor.*

- **gds\_l\_element\_t** **gds\_l\_cursor\_insert\_after** (**gds\_l\_cursor\_t** C, void \*VALUE)

*Insert a new element after a cursor.*

- **gds\_l\_element\_t** **gds\_l\_cursor\_insert\_before** (**gds\_l\_cursor\_t** C, void \*VALUE)

*Insert a new element before a cursor.*

- **gds\_l\_element\_t** **gds\_l\_cursor\_remove** (**gds\_l\_cursor\_t** C)

*Remove the element under a cursor.*

- **gds\_l\_element\_t** **gds\_l\_cursor\_remove\_after** (**gds\_l\_cursor\_t** C)

*Remove the element after a cursor.*

- **gdsl\_element\_t** **gdsl\_list\_cursor\_remove\_before** (**gdsl\_list\_cursor\_t** C)

*Remove the element before a cursor.*

- **gdsl\_list\_cursor\_t** **gdsl\_list\_cursor\_delete** (**gdsl\_list\_cursor\_t** C)

*Delete the element under a cursor.*

- **gdsl\_list\_cursor\_t** **gdsl\_list\_cursor\_delete\_after** (**gdsl\_list\_cursor\_t** C)

*Delete the element after a cursor.*

- **gdsl\_list\_cursor\_t** **gdsl\_list\_cursor\_delete\_before** (**gdsl\_list\_cursor\_t** C)

*Delete the element before the cursor of a list.*

### 4.11.3 Typedef Documentation

#### 4.11.3.1 typedef struct \_gdsl\_list\* **gdsl\_list\_t**

GDSL doubly-linked list type.

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

Definition at line 70 of file `gdsl_list.h`.

#### 4.11.3.2 typedef struct \_gdsl\_list\_cursor\* **gdsl\_list\_cursor\_t**

GDSL doubly-linked list cursor type.

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

Definition at line 78 of file `gdsl_list.h`.

### 4.11.4 Function Documentation

#### 4.11.4.1 **gdsl\_list\_t** **gdsl\_list\_alloc** ( const char \* *NAME*, **gdsl\_alloc\_func\_t** *ALLOC\_F*, **gdsl\_free\_func\_t** *FREE\_F* )

Create a new list.

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

- the default *ALLOC\_F* simply returns its argument
- the default *FREE\_F* does nothing

Note

Complexity:  $O(1)$



**Precondition**

nothing

**Parameters**

<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. 143)  
**gdsI\_list\_flush()** (p. 144)

**Examples:**

**examples/main\_list.c.**

**4.11.4.2 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. 142).

**Note**

Complexity:  $O(|L|)$

**Precondition**

L must be a valid gdsI\_list\_t

**Parameters**

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

See also

**gdsI\_list\_alloc()** (p. 142)

**gdsI\_list\_flush()** (p. 144)

Examples:

**examples/main\_list.c.**

#### 4.11.4.3 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. 142). 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. 142)

**gdsI\_list\_free()** (p. 143)

Examples:

**examples/main\_list.c.**

#### 4.11.4.4 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**

$L$	The list to get the name from
-----	-------------------------------

**Returns**

the name of the list  $L$ .

**See also**

**gdsl\_list\_set\_name()** (p. 147)

**Examples:**

**examples/main\_list.c.**

**4.11.4.5 `ulong gsdl_list_get_size ( const gsdl_list_t L )`**

Get the size of a list.

**Note**

Complexity:  $O(1)$

**Precondition**

$L$  must be a valid `gsdl_list_t`

**Parameters**

$L$	The list to get the size from
-----	-------------------------------

**Returns**

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

**Examples:**

**examples/main\_list.c.**

**4.11.4.6 `bool gsdl_list_is_empty ( const gsdl_list_t L )`**

Check if a list is empty.

**Note**

Complexity:  $O(1)$

**Precondition**

$L$  must be a valid `gdsi_list_t`

**Parameters**

$L$	The list to check
-----	-------------------

**Returns**

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

**Examples:**

**examples/main\_list.c.**

**4.11.4.7 `gdsi_element_t gdsi_list_get_head( const gdsi_list_t L )`**

Get the head of a list.

**Note**

Complexity:  $O(1)$

**Precondition**

$L$  must be a valid `gdsi_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**

**`gdsi_list_get_tail()`** (p. 147)

#### 4.11.4.8 `gdsl_element_t gsdl_list_get_tail( const gsdl_list_t L )`

Get the tail of a list.

##### Note

Complexity:  $O(1)$

##### Precondition

L must be a valid `gsdl_list_t`

##### Parameters

<code>L</code>	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

**`gsdl_list_get_head()`** (p. 146)

#### 4.11.4.9 `gsdl_list_t gsdl_list_set_name( gsdl_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 `gsdl_list_t`

##### Parameters

<code>L</code>	The list to change the name
<code>NEW_NAME</code>	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. 144)

**4.11.4.10 gdsl\_element\_t gdsl\_list\_insert\_head( gdsl\_list\_t L, void \* VALUE )**

Insert an element at the head of a list.

Allocate a new element E by calling L's ALLOC\_F function on VALUE. ALLOC\_F is the function pointer passed to **gdsl\_list\_alloc()** (p. 142). 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**

**gdsl\_list\_insert\_tail()** (p. 149)  
**gdsl\_list\_remove\_head()** (p. 149)  
**gdsl\_list\_remove\_tail()** (p. 150)  
**gdsl\_list\_remove()** (p. 151)

**Examples:**

**examples/main\_list.c.**

**4.11.4.11 `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. 142). 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. 148)  
**`gdsI_list_remove_head()`** (p. 149)  
**`gdsI_list_remove_tail()`** (p. 150)  
**`gdsI_list_remove()`** (p. 151)

**Examples:**

**`examples/main_list.c`.**

**4.11.4.12 `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 `gdsI_list_t`

**Parameters**

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

**Returns**

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

**See also**

**`gdsI_list_insert_head()`** (p. 148)  
**`gdsI_list_insert_tail()`** (p. 149)  
**`gdsI_list_remove_tail()`** (p. 150)  
**`gdsI_list_remove()`** (p. 151)

**4.11.4.13 `gdsI_element_t gdsI_list_remove_tail( gdsI_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 `gdsI_list_t`

**Parameters**

<b>L</b>	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. 148)  
**`gdsI_list_insert_tail()`** (p. 149)  
**`gdsI_list_remove_head()`** (p. 149)



**gdsl\_list\_remove()** (p. 151)

4.11.4.14 **gdsl\_element\_t** **gdsl\_list\_remove**( **gdsl\_list\_t** *L*, **gdsl\_compare\_func\_t** *COMP\_F*, const void \* *VALUE* )

Remove a particular element from a list.

Search into the list *L* for the first element *E* equal to *VALUE* by using *COMP\_F*. If *E* is found, it is removed from *L* and then returned.

**Note**

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

**Precondition**

*L* must be a valid **gdsl\_list\_t** & *COMP\_F* != NULL

**Parameters**

<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**

**gdsl\_list\_insert\_head()** (p. 148)

**gdsl\_list\_insert\_tail()** (p. 149)

**gdsl\_list\_remove\_head()** (p. 149)

**gdsl\_list\_remove\_tail()** (p. 150)

4.11.4.15 **gdsl\_list\_t** **gdsl\_list\_delete\_head**( **gdsl\_list\_t** *L* )

Delete the head of a list.

Remove the header element from the list *L* and deallocates it using the *FREE\_F* function passed to **gdsl\_list\_alloc()** (p. 142).

**Note**

Complexity:  $O(1)$

**Precondition**

L must be a valid `gdsl_list_t`

**Parameters**

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

**Returns**

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

**See also**

**gdsl\_list\_alloc()** (p. 142)  
`gdsl_list_destroy_tail()`  
`gdsl_list_destroy()`

**Examples:**

**examples/main\_list.c.**

**4.11.4.16 `gdsl_list_t gdsl_list_delete_tail( gdsl_list_t L )`**

Delete the tail of a list.

Remove the footer element from the list L and deallocates it using the `FREE_F` function passed to **gdsl\_list\_alloc()** (p. 142).

**Note**

Complexity:  $O(1)$

**Precondition**

L must be a valid `gdsl_list_t`

**Parameters**

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

**Returns**

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

See also

**gdsl\_list\_alloc()** (p. 142)  
**gdsl\_list\_destroy\_head()**  
**gdsl\_list\_destroy()**

Examples:

**examples/main\_list.c.**

4.11.4.17 **gdsl\_list\_t** **gdsl\_list\_delete**( **gdsl\_list\_t** *L*, **gdsl\_compare\_func\_t** *COMP\_F*,  
const void \* *VALUE* )

Delete a particular element from a list.

Search into the list *L* for the first element *E* equal to *VALUE* by using *COMP\_F*. If *E* is found, it is removed from *L* and deallocated using the *FREE\_F* function passed to **gdsl\_list\_alloc()** (p. 142).

Note

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

Precondition

*L* must be a valid **gdsl\_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

**gdsl\_list\_alloc()** (p. 142)  
**gdsl\_list\_destroy\_head()**  
**gdsl\_list\_destroy\_tail()**

Examples:

**examples/main\_list.c.**

4.11.4.18 **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. 154)  
**gdsI\_list\_search\_max()** (p. 155)  
**gdsI\_list\_search\_min()** (p. 156)

#### Examples:

**examples/main\_list.c.**

4.11.4.19 **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. 154)  
**`gdsi_list_search_max()`** (p. 155)  
**`gdsi_list_search_min()`** (p. 156)

**Examples:**

**`examples/main_list.c`**.

**4.11.4.20** `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` &  $COMP\_F \neq \text{NULL}$

**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. 154)  
**gdsI\_list\_search\_by\_position()** (p. 154)  
**gdsI\_list\_search\_min()** (p. 156)

**Examples:**

**examples/main\_list.c.**

#### 4.11.4.21 **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. 154)  
**gdsI\_list\_search\_by\_position()** (p. 154)  
**gdsI\_list\_search\_max()** (p. 155)

4.11.4.22 `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**

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

**Returns**

the sorted list L.

**Examples:**

`examples/main_list.c`.

4.11.4.23 `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 GDSI\_MAP\_STOP, then **`gdsi_list_map_forward()`** (p. 157) stops and returns its last examined element.

**Note**

Complexity:  $O(|L|)$

**Precondition**

L must be a valid `gdsi_list_t` & MAP\_F != NULL

**Parameters**

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

**Returns**

the first element for which MAP\_F returns GDSL\_MAP\_STOP.  
 NULL when the parsing is done.

**See also**

**gdsl\_list\_map\_backward()** (p. 158)

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

Parse a list from tail to head.

Parse all elements of the list L from tail to head. The MAP\_F function is called on each L's element with USER\_DATA argument. If MAP\_F returns GDSL\_MAP\_STOP then **gdsl\_list\_map\_backward()** (p. 158) 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 L's element
<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\_list\_map\_forward()** (p. 157)

**Examples:**

**examples/main\_list.c.**



4.11.4.25 `void gdsi_list_write( const gdsi_list_t L, gdsi_write_func_t WRITE_F, FILE * OUTPUT_FILE, void * USER_DATA )`

Write all the elements of a list to a file.

Write the elements of the list L to OUTPUT\_FILE, using WRITE\_F function. Additionnal USER\_DATA argument could be passed to WRITE\_F.

#### Note

Complexity:  $O(|L|)$

#### Precondition

L must be a valid gdsi\_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. 159)

**gdsi\_list\_dump()** (p. 160)

4.11.4.26 `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. 159)

**gdsi\_list\_dump()** (p. 160)

Examples:

**examples/main\_list.c.**

4.11.427 void **gdsi\_list\_dump**( const **gdsi\_list\_t** *L*, **gdsi\_write\_func\_t** *WRITE\_F*, FILE \* *OUTPUT\_FILE*, void \* *USER\_DATA* )

Dump the internal structure of a list to a file.

Dump the structure of the list *L* to *OUTPUT\_FILE*. If *WRITE\_F* != NULL, then uses - *WRITE\_F* to write L's elements to *OUTPUT\_FILE*. Additionnal *USER\_DATA* argument could be passed to *WRITE\_F*.

Note

Complexity:  $O(|L|)$

Precondition

*L* must be a valid **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. 159)

**gdsI\_list\_write\_xml()** (p. 159)

Examples:

**examples/main\_list.c.**

#### 4.11.4.28 **gdsI\_list\_cursor\_t gdsI\_list\_cursor\_alloc( const gdsI\_list\_t L )**

Create a new list cursor.

Note

Complexity:  $O(1)$

Precondition

L must be a valid gdsI\_list\_t

Parameters

<b>L</b>	The list on which the cursor is positioned.
----------	---

Returns

the newly allocated list cursor in case of success.

NULL in case of insufficient memory.

See also

**gdsI\_list\_cursor\_free()** (p. 161)

Examples:

**examples/main\_list.c.**

#### 4.11.4.29 **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. 161)

**Examples:**

**`examples/main_list.c`.**

**4.11.4.30 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. 162)

**Examples:**

**`examples/main_list.c`.**

**4.11.4.31 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**

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

**See also**

**`gdsI_list_cursor_move_to_head()`** (p. 162)

**4.11.4.32 `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**

C	The cursor to put on the element E
COMP_F	The comparison function to search for E
VALUE	The value used to compare list's elements with

**Returns**

the first founded element E in case it exists.  
NULL in case of element E is not found.

See also

**gdsI\_list\_cursor\_move\_to\_position()** (p. 164)

Examples:

**examples/main\_list.c.**

#### 4.11.4.33 **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.

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 gdsI\_list\_cursor\_t &  $POS > 0$  &  $POS \leq |L|$

Parameters

<i>C</i>	The cursor to put on the POS-th element
<i>POS</i>	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

**gdsI\_list\_cursor\_move\_to\_value()** (p. 163)

#### 4.11.4.34 **void gdsI\_list\_cursor\_step\_forward( gdsI\_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 `gdsf_list_cursor_t`

**Parameters**

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

**See also**

**`gdsf_list_cursor_step_backward()`** (p. 165)

**Examples:**

**`examples/main_list.c`.**

**4.11.4.35 void `gdsf_list_cursor_step_backward( gdsf_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 `gdsf_list_cursor_t`

**Parameters**

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

**See also**

**`gdsf_list_cursor_step_forward()`** (p. 164)

**Examples:**

**`examples/main_list.c`.**

**4.11.4.36 bool `gdsf_list_cursor_is_on_head( const gdsf_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. 166)

**4.11.4.37 `bool gdsl_list_cursor_is_on_tail( const gdsl_list_cursor_t C )`**

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

**Note**

Complexity:  $O(1)$

**Precondition**

C must be a valid `gdsl_list_cursor_t`

**Parameters**

C	The cursor to check
---	---------------------

**Returns**

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

**See also**

**`gdsl_list_cursor_is_on_head()`** (p. 165)



**4.11.4.38 bool gdsI\_list\_cursor\_has\_succ( const gdsI\_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. 167)

**4.11.4.39 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. 167)

4.11.4.40 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. 168) 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. 168)

4.11.4.41 **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. 168)

**Examples:**

**examples/main\_list.c.**

**4.11.4.42 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. 142). 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**

<b>C</b>	The cursor after which the new element must be inserted
<b>VALUE</b>	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. 170)

**gdsI\_list\_cursor\_remove\_after()** (p. 171)

**gdsI\_list\_cursor\_remove\_before()** (p. 172)

**Examples:**

**examples/main\_list.c.**

#### 4.11.4.43 `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. 142). 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 `gdsI_list_cursor_t`

##### Parameters

<code>C</code>	The cursor before which the new element must be inserted
<code>VALUE</code>	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_after()` (p. 169)  
`gdsI_list_cursor_remove_after()` (p. 171)  
`gdsI_list_cursor_remove_before()` (p. 172)

##### Examples:

`examples/main_list.c`.

#### 4.11.4.44 `gdsI_element_t gdsI_list_cursor_remove( gdsI_list_cursor_t C )`

Remove the element under a cursor.

##### Note

Complexity:  $O(1)$

**Precondition**

C must be a valid `gdsf_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**

**`gdsf_list_cursor_insert_after()`** (p. 169)  
**`gdsf_list_cursor_insert_before()`** (p. 170)  
**`gdsf_list_cursor_remove()`** (p. 170)  
**`gdsf_list_cursor_remove_before()`** (p. 172)

**4.11.4.45 `gdsf_element_t gdsf_list_cursor_remove_after( gdsf_list_cursor_t C )`**

Removec the element after a cursor.

**Note**

Complexity:  $O(1)$

**Precondition**

C must be a valid `gdsf_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. 169)  
**gdsI\_list\_cursor\_insert\_before()** (p. 170)  
**gdsI\_list\_cursor\_remove()** (p. 170)  
**gdsI\_list\_cursor\_remove\_before()** (p. 172)

#### 4.11.4.46 **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 **gdsI\_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

**gdsI\_list\_cursor\_insert\_after()** (p. 169)  
**gdsI\_list\_cursor\_insert\_before()** (p. 170)  
**gdsI\_list\_cursor\_remove()** (p. 170)  
**gdsI\_list\_cursor\_remove\_after()** (p. 171)

#### 4.11.4.47 **gdsI\_list\_cursor\_t gdsI\_list\_cursor\_delete( gdsI\_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 **gdsI\_list\_alloc()** (p. 142).

Complexity:  $O(1)$

Precondition

C must be a valid **gdsI\_list\_cursor\_t**

## Parameters

<b>C</b>	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

**gdsI\_list\_cursor\_delete\_before()** (p. 173)  
**gdsI\_list\_cursor\_delete\_after()** (p. 173)

4.11.4.48 **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. 142).

Complexity: O( 1 )

## Precondition

C must be a valid gdsI\_list\_cursor\_t

## Parameters

<b>C</b>	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. 172)  
**gdsI\_list\_cursor\_delete\_before()** (p. 173)

## Examples:

**examples/main\_list.c.**

4.11.4.49 **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. 142).

**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. 172)  
**`gdsi_list_cursor_delete_after()`** (p. 173)



## 4.12 Various macros module

### 4.12.1

#### 4.12.2 Copyright

Copyright (C) 1998-2017 Nicolas Darnis <ndarnis@free.fr>.

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

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

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

#### Defines

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

#### 4.12.3 Define Documentation

##### 4.12.3.1 `#define GDSDL_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. 176)

Definition at line 72 of file gdsl\_macros.h.

4.12.3.2 **#define GDSL\_MIN( X, Y ) (X>Y?Y:X)**

Give the lowest number of two numbers.

**Note**

Complexity:  $O(1)$

**Precondition**

X & Y must be basic scalar C types

**Parameters**

X	First scalar variable
Y	Second scalar variable

**Returns**

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

**See also**

**GDSL\_MAX()** (p. 175)

Definition at line 89 of file gdsl\_macros.h.

## 4.13 Permutation manipulation module

### 4.13.1

#### 4.13.2 Copyright

Copyright (C) 1998-2017 Nicolas Darnis <ndarnis@free.fr>.

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

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

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

#### Typedefs

- typedef struct gds\_l\_perm \* **gds\_l\_perm\_t**  
*GDSDL permutation type.*
- typedef void(\* **gds\_l\_perm\_write\_func\_t**)(ulong E, FILE \*OUTPUT\_FILE, **gds\_l\_perm\_location\_t** POSITION, void \*USER\_DATA)  
*GDSDL permutation write function type.*
- typedef struct gds\_l\_perm\_data \* **gds\_l\_perm\_data\_t**

#### Enumerations

- enum **gds\_l\_perm\_position\_t** { **GDSDL\_PERM\_POSITION\_FIRST** = 1, **GDSDL\_PERM\_POSITION\_LAST** = 2 }  
*This type is for gds\_l\_perm\_write\_func\_t.*

#### Functions

- **gds\_l\_perm\_t gds\_l\_perm\_alloc** (const char \*NAME, const ulong N)  
*Create a new permutation.*
- void **gds\_l\_perm\_free** (**gds\_l\_perm\_t** P)  
*Destroy a permutation.*
- **gds\_l\_perm\_t gds\_l\_perm\_copy** (const **gds\_l\_perm\_t** P)  
*Copy a permutation.*
- const char \* **gds\_l\_perm\_get\_name** (const **gds\_l\_perm\_t** P)  
*Get the name of a permutation.*
- ulong **gds\_l\_perm\_get\_size** (const **gds\_l\_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.*
- **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.*

### 4.13.3 Typedef Documentation

#### 4.13.3.1 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 69 of file gdsI\_perm.h.

#### 4.13.3.2 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 <b>gdsI_perm_position_t</b> values to indicate where E is located into the <b>gdsI_perm_t</b> mapped.
<i>USER_DATA</i>	User's datas

Definition at line 93 of file gdsI\_perm.h.

#### 4.13.3.3 typedef struct gdsI\_perm\_data\* gdsI\_perm\_data\_t

Definition at line 99 of file gdsI\_perm.h.

### 4.13.4 Enumeration Type Documentation

#### 4.13.4.1 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 74 of file gdsI\_perm.h.

### 4.13.5 Function Documentation

#### 4.13.5.1 `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* with 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. 180)  
`gdsI_perm_copy()` (p. 181)

##### Examples:

`examples/main_bstree.c`, `examples/main_list.c`, `examples/main_llbstree.c`,  
`examples/main_perm.c`, and `examples/main_rbtrees.c`.

#### 4.13.5.2 `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

$P$	The permutation to destroy
-----	----------------------------

## See also

**gdsi\_perm\_alloc()** (p. 180)

**gdsi\_perm\_copy()** (p. 181)

## Examples:

**examples/main\_bstree.c**, **examples/main\_list.c**, **examples/main\_llbstree.c**,  
**examples/main\_perm.c**, and **examples/main\_rbtrees.c**.

4.13.5.3 **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

$P$	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. 180)

**gdsi\_perm\_free** (p. 180)

#### 4.13.5.4 `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. 186)

#### 4.13.5.5 `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

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



**Returns**

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

**See also**

**gdsI\_perm\_get\_element()** (p. 183)

**gdsI\_perm\_get\_elements\_array()** (p. 183)

**4.13.5.6 `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**

$P$	The permutation to use.
$INDIX$	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. 182)

**gdsI\_perm\_get\_elements\_array()** (p. 183)

**Examples:**

**examples/main\_bstree.c**, **examples/main\_list.c**, **examples/main\_llbtree.c**,  
and **examples/main\_rbtrees.c**.

**4.13.5.7 `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 datas from.
----------	------------------------------------

**Returns**

the values array of the permutation P.

**See also**

**`gdsI_perm_get_element()`** (p. 183)  
**`gdsI_perm_set_elements_array()`** (p. 188)

**4.13.5.8 `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.

**Examples:**

**`examples/main_perm.c`**.

**4.13.5.9 `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. 185)

**Examples:**

**`examples/main_perm.c`**.

**4.13.5.10 `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 `gdsi_perm_t`

**Parameters**

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

**Returns**

the number of cycles into the canonical permutation P.

See also

**gdsI\_perm\_linear\_cycles\_count()** (p. 184)

Examples:

**examples/main\_perm.c.**

**4.13.5.11 gdsI\_perm\_t gdsI\_perm\_set\_name ( gdsI\_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 gdsI\_perm\_t

Parameters

<i>P</i>	The permutation to change the name
NEW_NAME	The new name of <i>P</i>

Returns

the modified permutation in case of success.

NULL in case of insufficient memory.

See also

**gdsI\_perm\_get\_name()** (p. 182)

**4.13.5.12 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. 187)

**4.13.5.13 `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. 186)

#### 4.13.5.14 `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. 183)

#### 4.13.5.15 `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 RESULT without modifying ALPHA and BETA.

##### Note

Complexity:  $O(|\text{RESULT}|)$

##### Precondition

RESULT, ALPHA and BETA must be valids `gdsi_perm_t` &  $|\text{RESULT}| == |\text{ALPHA}|$   
 $== |\text{BETA}|$

##### 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.

#### 4.13.5.16 `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. 189)

**Examples:**

`examples/main_perm.c`.

#### 4.13.5.17 `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. 189)

#### Examples:

**`examples/main_perm.c`**.

#### 4.13.5.18 `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. 191)

**Examples:**

**examples/main\_perm.c.**

**4.13.5.19 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. 190)

**Examples:**

**examples/main\_perm.c.**

**4.13.5.20 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 `gdsl_perm_t`

**Parameters**

<i>P</i>	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.

**Examples:**

**examples/main\_bstree.c**, **examples/main\_list.c**, **examples/main\_llbstree.c**,  
**examples/main\_perm.c**, and **examples/main\_rbstree.c**.

#### 4.13.5.21 `gdsl_element_t* gdsl_perm_apply_on_array( gdsl_element_t* V, const gdsl_perm_t P )`

Apply a permutation on to a vector.

**Note**

Complexity:  $O(|P|)$

**Precondition**

P must be a valid `gdsl_perm_t` &  $|P| == |V|$

**Parameters**

<i>V</i>	The vector/array to reorder according to P
<i>P</i>	The permutation to use to reorder V

**Returns**

the reordered array V according to the permutation P in case of success.  
 NULL in case of insufficient memory.

**Examples:**

**examples/main\_perm.c**.

4.13.5.22 `void 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

`gdsI_perm_write_xml()` (p. 193)

`gdsI_perm_dump()` (p. 194)

#### Examples:

`examples/main_perm.c`.

4.13.5.23 `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.

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. Additional 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 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 <i>WRITE_F</i> .

**See also**

**`gdsI_perm_write()`** (p. 193)  
**`gdsI_perm_dump()`** (p. 194)

**4.13.5.24** `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 <i>WRITE_F</i> .

**See also**

**`gdsI_perm_write()`** (p. 193)  
**`gdsI_perm_write_xml()`** (p. 193)

## 4.14 Queue manipulation module

### 4.14.1

#### 4.14.2 Copyright

Copyright (C) 1998-2017 Nicolas Darnis <ndarnis@free.fr>.

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

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

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

#### Typedefs

- typedef struct \_gdsl\_queue \* **gdsl\_queue\_t**  
*GDSDL 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 gsdl\_queue\_insert** (gsdl\_queue\_t Q, void \*VALUE)  
*Insert an element in a queue (PUT).*
- **gsdl\_element\_t gsdl\_queue\_remove** (gsdl\_queue\_t Q)  
*Remove an element from a queue (GET).*
- **gsdl\_element\_t gsdl\_queue\_search** (const gsdl\_queue\_t Q, gsdl\_compare\_func\_t COMP\_F, void \*VALUE)  
*Search for a particular element in a queue.*
- **gsdl\_element\_t gsdl\_queue\_search\_by\_position** (const gsdl\_queue\_t Q, ulong POS)  
*Search for an element by its position in a queue.*
- **gsdl\_element\_t gsdl\_queue\_map\_forward** (const gsdl\_queue\_t Q, gsdl\_map\_func\_t MAP\_F, void \*USER\_DATA)  
*Parse a queue from head to tail.*
- **gsdl\_element\_t gsdl\_queue\_map\_backward** (const gsdl\_queue\_t Q, gsdl\_map\_func\_t MAP\_F, void \*USER\_DATA)  
*Parse a queue from tail to head.*
- void **gsdl\_queue\_write** (const gsdl\_queue\_t Q, gsdl\_write\_func\_t WRITE\_F, FILE \*OUTPUT\_FILE, void \*USER\_DATA)  
*Write all the elements of a queue to a file.*
- void **gsdl\_queue\_write\_xml** (const gsdl\_queue\_t Q, gsdl\_write\_func\_t WRITE\_F, FILE \*OUTPUT\_FILE, void \*USER\_DATA)  
*Write the content of a queue to a file into XML.*
- void **gsdl\_queue\_dump** (const gsdl\_queue\_t Q, gsdl\_write\_func\_t WRITE\_F, FILE \*OUTPUT\_FILE, void \*USER\_DATA)  
*Dump the internal structure of a queue to a file.*

### 4.14.3 Typedef Documentation

#### 4.14.3.1 typedef struct \_gsdl\_queue\* gsdl\_queue\_t

GDSDL 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 74 of file gsdl\_queue.h.

### 4.14.4 Function Documentation

#### 4.14.4.1 gsdl\_queue\_t gsdl\_queue\_alloc( const char \* NAME, gsdl\_alloc\_func\_t ALLOC\_F, gsdl\_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. 197)  
**`gdsI_queue_flush()`** (p. 198)

**Examples:**

**`examples/main_queue.c`.**

**4.14.4.2 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. 196). 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

**gdsl\_queue\_alloc()** (p. 196)

**gdsl\_queue\_flush()** (p. 198)

## Examples:

**examples/main\_queue.c.**

#### 4.14.4.3 void **gdsl\_queue\_flush**( **gsdl\_queue\_t** Q )

Flush a queue.

Deallocate all the elements of the queue Q by calling Q's **FREE\_F** function passed to **gsdl\_queue\_alloc()**. Q is not deallocated itself and Q's name is not modified.

## Note

Complexity:  $O(|Q|)$

## Precondition

Q must be a valid **gsdl\_queue\_t**

## Parameters

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

## See also

**gsdl\_queue\_alloc()** (p. 196)

**gsdl\_queue\_free()** (p. 197)

## Examples:

**examples/main\_queue.c.**

#### 4.14.4.4 const char\* **gsdl\_queue\_get\_name**( const **gsdl\_queue\_t** Q )

Getsthe name of a queue.

## Note

Complexity:  $O(1)$



**Precondition**

Q must be a valid `gdsl_queue_t`

**Postcondition**

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

**Parameters**

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

**Returns**

the name of the queue Q.

**See also**

**`gdsl_queue_set_name()`** (p. 201)

**Examples:**

**`examples/main_queue.c`.**

**4.14.4.5 `ulong gsdl_queue_get_size( const gsdl_queue_t Q )`**

Get the size of a queue.

**Note**

Complexity:  $O(1)$

**Precondition**

Q must be a valid `gsdl_queue_t`

**Parameters**

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

**Returns**

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

#### 4.14.4.6 `bool 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.

##### Examples:

`examples/main_queue.c.`

#### 4.14.4.7 `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. 201)

Examples:

**examples/main\_queue.c.**

#### 4.14.4.8 **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. 200)

Examples:

**examples/main\_queue.c.**

#### 4.14.4.9 **gdsI\_queue\_t gdsI\_queue\_set\_name( gdsI\_queue\_t Q, const char \* NEW\_NAME )**

Set the name of a queue.

Change the previous name of the queue Q to a copy of NEW\_NAME.

Note

Complexity:  $O(1)$

**Precondition**

Q must be a valid `gdsl_queue_t`

**Parameters**

Q	The queue to change the name
NEW_NAME	The new name of Q

**Returns**

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

**See also**

**`gdsl_queue_get_name()`** (p. 198)

**4.14.4.10 `gdsl_element_t` `gdsl_queue_insert( gdsl_queue_t Q, void * VALUE )`**

Insert an element in a queue (PUT).

Allocate a new element E by calling Q's `ALLOC_F` function on VALUE. `ALLOC_F` is the function pointer passed to **`gdsl_queue_alloc()`** (p. 196). The new element E is then inserted at the header position of the queue Q.

**Note**

Complexity:  $O(1)$

**Precondition**

Q must be a valid `gdsl_queue_t`

**Parameters**

Q	The queue to insert in
VALUE	The value used to make the new element to insert into Q

**Returns**

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

See also

**gdsI\_queue\_remove()** (p. 203)

Examples:

**examples/main\_queue.c.**

#### 4.14.4.11 **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. 202)

Examples:

**examples/main\_queue.c.**

#### 4.14.4.12 **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. 204)

#### 4.14.4.13 `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

**gdsI\_queue\_search()** (p. 203)

Examples:

**examples/main\_queue.c.**

4.14.4.14 **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.

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 **gdsI\_queue\_map\_forward()** (p. 205) stops and returns its last examined element.

Note

Complexity:  $O(|Q|)$

Precondition

Q must be a valid gdsI\_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

**gdsI\_queue\_map\_backward()** (p. 206)

Examples:

**examples/main\_queue.c.**

#### 4.14.4.15 `gdsl_element_t gdsl_queue_map_backward( const gdsl_queue_t Q, gdsl_map_func_t MAP_F, void * USER_DATA )`

Parse a queue from tail to head.

Parse all elements of the queue Q from tail to head. The MAP\_F function is called on each Q's element with USER\_DATA argument. If MAP\_F returns GDSL\_MAP\_STOP, then **gdsl\_queue\_map\_backward()** (p. 206) 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. Returns NULL when the parsing is done.

##### See also

**gdsl\_queue\_map\_forward()** (p. 205)

#### 4.14.4.16 `void gdsl_queue_write( const gdsl_queue_t Q, gdsl_write_func_t WRITE_F, FILE * OUTPUT_FILE, void * USER_DATA )`

Write all the elements of a queue to a file.

Write the elements of the queue Q to OUTPUT\_FILE, using WRITE\_F function. - Additionnal USER\_DATA argument could be passed to WRITE\_F.

##### Note

Complexity:  $O(|Q|)$

##### Precondition

Q must be a valid `gdsl_queue_t` & OUTPUT\_FILE != NULL & WRITE\_F != NULL



## Parameters

<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

**gdsi\_queue\_write\_xml()** (p. 207)

**gdsi\_queue\_dump()** (p. 208)

4.14.4.17 `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 Q's elements.
<i>USER_DATA</i>	User's datas passed to <i>WRITE_F</i> .

## See also

**gdsi\_queue\_write()** (p. 206)

**gdsi\_queue\_dump()** (p. 208)

## Examples:

**examples/main\_queue.c.**

4.14.4.18 `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 gdsI\_queue\_t & OUTPUT\_FILE != NULL

#### Parameters

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

#### See also

`gdsI_queue_write()` (p. 206)  
`gdsI_queue_write_xml()` (p. 207)

#### Examples:

`examples/main_queue.c`.

## 4.15 Red-black tree manipulation module

### 4.15.1

### 4.15.2 Copyright

Copyright (C) 1998-2017 Nicolas Darnis <ndarnis@free.fr>.

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

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

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

### Typedefs

- typedef struct gds\_l\_rbtrees \* **gds\_l\_rbtrees\_t**

### Functions

- **gds\_l\_rbtrees\_t gds\_l\_rbtrees\_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 red-black tree.*
- void **gds\_l\_rbtrees\_free** (**gds\_l\_rbtrees\_t** T)  
*Destroy a red-black tree.*
- void **gds\_l\_rbtrees\_flush** (**gds\_l\_rbtrees\_t** T)  
*Flush a red-black tree.*
- char \* **gds\_l\_rbtrees\_get\_name** (const **gds\_l\_rbtrees\_t** T)  
*Get the name of a red-black tree.*
- bool **gds\_l\_rbtrees\_is\_empty** (const **gds\_l\_rbtrees\_t** T)  
*Check if a red-black tree is empty.*
- **gds\_l\_element\_t gds\_l\_rbtrees\_get\_root** (const **gds\_l\_rbtrees\_t** T)  
*Get the root of a red-black tree.*
- **ulong gds\_l\_rbtrees\_get\_size** (const **gds\_l\_rbtrees\_t** T)  
*Get the size of a red-black tree.*
- **ulong gds\_l\_rbtrees\_height** (const **gds\_l\_rbtrees\_t** T)  
*Get the height of a red-black tree.*
- **gds\_l\_rbtrees\_t gds\_l\_rbtrees\_set\_name** (**gds\_l\_rbtrees\_t** T, const char \*NEW\_NAME)  
*Set the name of a red-black tree.*

- **gds\_l\_element\_t gds\_l\_rbtrees\_insert** (gds\_l\_rbtrees\_t T, void \*VALUE, int \*RESULT)  
*Insert an element into a red-black tree if it's not found or return it.*
- **gds\_l\_element\_t gds\_l\_rbtrees\_remove** (gds\_l\_rbtrees\_t T, void \*VALUE)  
*Remove an element from a red-black tree.*
- **gds\_l\_rbtrees\_t gds\_l\_rbtrees\_delete** (gds\_l\_rbtrees\_t T, void \*VALUE)  
*Delete an element from a red-black tree.*
- **gds\_l\_element\_t gds\_l\_rbtrees\_search** (const gds\_l\_rbtrees\_t T, gds\_l\_compare\_func\_t COMP\_F, void \*VALUE)  
*Search for a particular element into a red-black tree.*
- **gds\_l\_element\_t gds\_l\_rbtrees\_map\_prefix** (const gds\_l\_rbtrees\_t T, gds\_l\_map\_func\_t MAP\_F, void \*USER\_DATA)  
*Parse a red-black tree in prefixed order.*
- **gds\_l\_element\_t gds\_l\_rbtrees\_map\_infix** (const gds\_l\_rbtrees\_t T, gds\_l\_map\_func\_t MAP\_F, void \*USER\_DATA)  
*Parse a red-black tree in infix order.*
- **gds\_l\_element\_t gds\_l\_rbtrees\_map\_postfix** (const gds\_l\_rbtrees\_t T, gds\_l\_map\_func\_t MAP\_F, void \*USER\_DATA)  
*Parse a red-black tree in postfix order.*
- void **gds\_l\_rbtrees\_write** (const gds\_l\_rbtrees\_t T, gds\_l\_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 **gds\_l\_rbtrees\_write\_xml** (const gds\_l\_rbtrees\_t T, gds\_l\_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 **gds\_l\_rbtrees\_dump** (const gds\_l\_rbtrees\_t T, gds\_l\_write\_func\_t WRITE\_F, FILE \*OUTPUT\_FILE, void \*USER\_DATA)  
*Dump the internal structure of a red-black tree to a file.*

### 4.15.3 Typedef Documentation

#### 4.15.3.1 typedef struct gds\_l\_rbtrees\* gds\_l\_rbtrees\_t

GDSL red-black tree type.

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

Definition at line 71 of file gds\_l\_rbtrees.h.

### 4.15.4 Function Documentation

#### 4.15.4.1 gds\_l\_rbtrees\_t gds\_l\_rbtrees\_alloc ( const char \* NAME, gds\_l\_alloc\_func\_t ALLOC\_F, gds\_l\_free\_func\_t FREE\_F, gds\_l\_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\_rbtrees\_free()** (p. 211)  
**gdsI\_rbtrees\_flush()** (p. 212)

**Examples:**

**examples/main\_rbtrees.c.**

**4.15.4.2 void gdsI\_rbtrees\_free( gdsI\_rbtrees\_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\_rbtrees\_alloc()** (p. 210). The name of T is deallocated and T is deallocated itself too.

**Note**

Complexity:  $O(|T|)$

**Precondition**

T must be a valid `gdsi_rbtrees_t`

**Parameters**

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

**See also**

**`gdsi_rbtrees_alloc()`** (p. 210)

**`gdsi_rbtrees_flush()`** (p. 212)

**Examples:**

**`examples/main_rbtrees.c`**.

**4.15.4.3 void `gdsi_rbtrees_flush( gdsi_rbtrees_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_rbtrees_alloc()`** (p. 210). 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_rbtrees_t`

**See also**

**`gdsi_rbtrees_alloc()`** (p. 210)

**`gdsi_rbtrees_free()`** (p. 211)

**Examples:**

**`examples/main_rbtrees.c`**.

#### 4.15.4.4 `char* gdsi_rbtrees_get_name( const gdsi_rbtrees_t T )`

Get the name of a red-black tree.

##### Note

Complexity:  $O(1)$

##### Precondition

T must be a valid `gdsi_rbtrees_t`

##### Postcondition

The returned string MUST NOT be freed.

##### Parameters

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

##### Returns

the name of the red-black tree T.

##### See also

**`gdsi_rbtrees_set_name()`** (p. 215)

#### 4.15.4.5 `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

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

**Returns**

TRUE if the red-black tree T is empty.  
FALSE if the red-black tree T is not empty.

**Examples:**

**examples/main\_rbtrees.c.**

**4.15.4.6 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**

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

**Returns**

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

**Examples:**

**examples/main\_rbtrees.c.**

**4.15.4.7 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**

<i>T</i>	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()`

**Examples:**

**examples/main\_rbtrees.c.**

**4.15.4.8 `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**

$T$	The red-black tree to compute the height from
-----	---

**Returns**

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

**See also**

**`gdsI_rbtrees_get_size()`** (p. 214)

**Examples:**

**examples/main\_rbtrees.c.**

**4.15.4.9 `gdsI_rbtrees_t gdsI_rbtrees_set_name( gdsI_rbtrees_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_rbtrees_t`

**Parameters**

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

**Returns**

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

**See also**

**`gdsI_rbtrees_get_name()`** (p. 213)

#### 4.15.4.10 `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.

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_rbtrees_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_rbtrees_alloc` and is inserted and then returned.

**Note**

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

**Precondition**

T must be a valid `gdsI_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. 217)

**gdsl\_rbtrees\_delete()** (p. 218)

**Examples:**

**examples/main\_rbtrees.c.**

**4.15.4.11 gsdl\_element\_t gsdl\_rbtrees\_remove( gsdl\_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 **gsdl\_rbtrees\_alloc()** (p. 210). If E is found, it is removed from T and then returned.

**Note**

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

**Precondition**

T must be a valid gsdl\_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**

**gsdl\_rbtrees\_insert()** (p. 216)

**gsdl\_rbtrees\_delete()** (p. 218)

#### 4.15.4.12 `gdsi_rbtrees_t gdsi_rbtrees_delete( gdsi_rbtrees_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_rbtrees_alloc()`** (p. 210). If E is found, it is removed from T and E is deallocated using T's FREE\_F function passed to **`gdsi_rbtrees_alloc()`** (p. 210), then T is returned.

##### Note

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

##### Precondition

T must be a valid `gdsi_rbtrees_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_rbtrees_insert()`** (p. 216)  
**`gdsi_rbtrees_remove()`** (p. 217)

##### Examples:

**`examples/main_rbtrees.c`**.

#### 4.15.4.13 `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.

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_rbtrees_alloc()`** (p. 210) is used.

##### Note

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

**Precondition**

T must be a valid `gdsl_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**

**`gdsl_rbtrees_insert()`** (p. 216)  
**`gdsl_rbtrees_remove()`** (p. 217)  
**`gdsl_rbtrees_delete()`** (p. 218)

**Examples:**

**`examples/main_rbtrees.c`.**

#### 4.15.4.14 `gdsl_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.

Parse all nodes of the red-black tree T in prefixed order. The MAP\_F function is called on the element contained in each node with the USER\_DATA argument. If MAP\_F returns GDSL\_MAP\_STOP, then **`gsdl_rbtrees_map_prefix()`** (p. 219) 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**

**gdsl\_rbtrees\_map\_infix()** (p. 220)  
**gdsl\_rbtrees\_map\_postfix()** (p. 221)

**Examples:**

**examples/main\_rbtrees.c.**

**4.15.4.15** **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. 220) 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. 219)  
**gsdl\_rbtrees\_map\_postfix()** (p. 221)

Examples:

**examples/main\_rbtrees.c.**

4.15.4.16 **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. 221) 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. 219)

**gdsI\_rbtrees\_map\_infix()** (p. 220)

Examples:

**examples/main\_rbtrees.c.**

4.15.4.17 **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 *gdsI\_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 <i>T</i> 's elements.
<i>USER_DATA</i>	User's datas passed to <i>WRITE_F</i> .

#### See also

***gdsI\_rbtrees\_write\_xml()*** (p. 222)

***gdsI\_rbtrees\_dump()*** (p. 223)

#### Examples:

**examples/main\_rbtrees.c.**

**4.15.4.18** `void gdsI_rbtrees_write_xml( const gdsI_rbtrees_t T, gdsI_write_func_t  
WRITE_F, FILE * OUTPUT_FILE, void * USER_DATA )`

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

Write the nodes elements of the red-black tree *T* to *OUTPUT\_FILE*, into XML language. If *WRITE\_F* != NULL, then use *WRITE\_F* to write *T*'s nodes elements to *OUTPUT\_FILE*. Additionnal *USER\_DATA* argument could be passed to *WRITE\_F*.

#### Note

Complexity:  $O(|T|)$

#### Precondition

*T* must be a valid *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 T's elements.
<i>USER_DATA</i>	User's datas passed to WRITE_F.

See also

**gdsI\_rbtrees\_write()** (p. 221)  
**gdsI\_rbtrees\_dump()** (p. 223)

Examples:

**examples/main\_rbtrees.c.**

**4.15.4.19** 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*. Additional *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 T's elements.
<i>USER_DATA</i>	User's datas passed to WRITE_F.

See also

**gdsi\_rbtrees\_write()** (p. 221)

**gdsi\_rbtrees\_write\_xml()** (p. 222)

Examples:

**examples/main\_rbtrees.c.**

## 4.16 Sort module

### 4.16.1

#### 4.16.2 Copyright

Copyright (C) 1998-2017 Nicolas Darnis <ndarnis@free.fr>.

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

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

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

### Functions

- void **gdsl\_sort** (**gdsl\_element\_t** \*T, **ulong** N, **const gdsl\_compare\_func\_t** COMP\_F)

*Sort an array in place.*

#### 4.16.3 Function Documentation

4.16.3.1 void **gdsl\_sort** ( **gdsl\_element\_t** \* T, **ulong** N, **const gdsl\_compare\_func\_t** COMP\_F )

Sort an array in place.

Sort the array T in place. The function COMP\_F is used to compare T's elements and must be user-defined.

#### Note

Complexity:  $O(N \log(N))$

#### Precondition

$N == |T|$  &  $T \neq \text{NULL}$  &  $\text{COMP\_F} \neq \text{NULL}$  & for all  $i \leq N$ :  $\text{sizeof}(T[i]) == \text{sizeof}(\text{gdsl\_element\_t})$

#### Parameters

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

Examples:

`examples/main_sort.c.`

## 4.17 Stack manipulation module

### 4.17.1

#### 4.17.2 Copyright

Copyright (C) 1998-2017 Nicolas Darnis <ndarnis@free.fr>.

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

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

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

#### 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 gsdl\_stack\_set\_name** (gsdl\_stack\_t S, const char \*NEW\_NAME)

*Set the name of a stack.*

- void **gsdl\_stack\_set\_growing\_factor** (gsdl\_stack\_t S, ulong G)

*Set the growing factor of a stack.*

- **gsdl\_element\_t gsdl\_stack\_insert** (gsdl\_stack\_t S, void \*VALUE)

*Insert an element in a stack (PUSH).*

- **gsdl\_element\_t gsdl\_stack\_remove** (gsdl\_stack\_t S)

*Remove an element from a stack (POP).*

- **gsdl\_element\_t gsdl\_stack\_search** (const gsdl\_stack\_t S, gsdl\_compare\_func\_t COMP\_F, void \*VALUE)

*Search for a particular element in a stack.*

- **gsdl\_element\_t gsdl\_stack\_search\_by\_position** (const gsdl\_stack\_t S, ulong POS)

*Search for an element by its position in a stack.*

- **gsdl\_element\_t gsdl\_stack\_map\_forward** (const gsdl\_stack\_t S, gsdl\_map\_func\_t MAP\_F, void \*USER\_DATA)

*Parse a stack from bottom to top.*

- **gsdl\_element\_t gsdl\_stack\_map\_backward** (const gsdl\_stack\_t S, gsdl\_map\_func\_t MAP\_F, void \*USER\_DATA)

*Parse a stack from top to bottom.*

- void **gsdl\_stack\_write** (const gsdl\_stack\_t S, gsdl\_write\_func\_t WRITE\_F, FILE \*OUTPUT\_FILE, void \*USER\_DATA)

*Write all the elements of a stack to a file.*

- void **gsdl\_stack\_write\_xml** (gsdl\_stack\_t S, gsdl\_write\_func\_t WRITE\_F, FILE \*OUTPUT\_FILE, void \*USER\_DATA)

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

- void **gsdl\_stack\_dump** (gsdl\_stack\_t S, gsdl\_write\_func\_t WRITE\_F, FILE \*OUTPUT\_FILE, void \*USER\_DATA)

*Dump the internal structure of a stack to a file.*

### 4.17.3 Typedef Documentation

#### 4.17.3.1 typedef struct \_gsdl\_stack\* gsdl\_stack\_t

GDSDL 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 72 of file gsdl\_stack.h.

#### 4.17.4 Function Documentation

4.17.4.1 `gdsl_stack_t` `gdsl_stack_alloc( const char * NAME, gdsl_alloc_func_t ALLOC_F, gdsl_free_func_t FREE_F )`

Create a new stack.

Allocate a new stack data structure which name is set to a copy of *NAME*. The functions pointers *ALLOC\_F* and *FREE\_F* could be used to respectively, alloc and free elements in the stack. These pointers could be set to NULL to use the default ones:

- the default *ALLOC\_F* simply returns its argument
- the default *FREE\_F* does nothing

##### Note

Complexity:  $O(1)$

##### Precondition

nothing.

##### Parameters

<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

`gdsl_stack_free()` (p. 229)  
`gdsl_stack_flush()` (p. 230)

##### Examples:

`examples/main_stack.c`.

4.17.4.2 `void` `gdsl_stack_free( gdsl_stack_t S )`

Destroy a stack.

Deallocate all the elements of the stack *S* by calling *S*'s *FREE\_F* function passed to `gdsl_stack_alloc()` (p. 229). The name of *S* is deallocated and *S* is deallocated itself too.

**Note**

Complexity:  $O(|S|)$

**Precondition**

S must be a valid `gdsl_stack_t`

**Parameters**

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

**See also**

**`gdsl_stack_alloc()`** (p. 229)

**`gdsl_stack_flush()`** (p. 230)

**Examples:**

**`examples/main_stack.c`**.

**4.17.4.3 void `gdsl_stack_flush( gdsl_stack_t S )`**

Flush a stack.

Deallocate all the elements of the stack S by calling S's `FREE_F` function passed to **`gdsl_stack_alloc()`** (p. 229). S is not deallocated itself and S's name is not modified.

**Note**

Complexity:  $O(|S|)$

**Precondition**

S must be a valid `gdsl_stack_t`

**Parameters**

S	The stack to flush
---	--------------------

**See also**

**`gdsl_stack_alloc()`** (p. 229)

**`gdsl_stack_free()`** (p. 229)

**Examples:**

**`examples/main_stack.c`**.



#### 4.17.4.4 `const char* gdsi_stack_get_name( const gdsi_stack_t S )`

Getsthe 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

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

##### Returns

the name of the stack S.

##### See also

**`gdsi_stack_set_name()`** (p. 234)

##### Examples:

**`examples/main_stack.c`**.

#### 4.17.4.5 `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  $|S|$ ).

**4.17.4.6 `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. 235).

**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. 235)

**`gdsi_stack_set_growing_factor()`** (p. 235)

**4.17.4.7 `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.

## Examples:

**examples/main\_stack.c.**

4.17.4.8 `gdsl_element_t gdsl_stack_get_top( const gdsl_stack_t S )`

Get the top of a stack.

## Note

Complexity:  $O(1)$

## Precondition

S must be a valid `gdsl_stack_t`

## Parameters

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

## Returns

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

## See also

**`gdsl_stack_get_bottom()`** (p. 233)

## Examples:

**examples/main\_stack.c.**

4.17.4.9 `gdsl_element_t gdsl_stack_get_bottom( const gdsl_stack_t S )`

Get the bottom of a stack.

**Note**

Complexity:  $O(1)$

**Precondition**

S must be a valid `gdsl_stack_t`

**Parameters**

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

**Returns**

the element contained at the bottom position of the stack S if S is not empty. The returned element is not removed from S.  
NULL if the stack S is empty.

**See also**

**`gdsl_stack_get_top()`** (p. 233)

#### 4.17.4.10 `gdsl_stack_t gdsl_stack_set_name ( gdsl_stack_t S, const char * NEW_NAME )`

Set the name of a stack.

Change the previous name of the stack S to a copy of NEW\_NAME.

**Note**

Complexity:  $O(1)$

**Precondition**

S must be a valid `gdsl_stack_t`

**Parameters**

S	The stack to change the name
NEW_NAME	The new name of S

**Returns**

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

See also

**gdsl\_stack\_get\_name()** (p. 231)

#### 4.17.4.11 void **gdsl\_stack\_set\_growing\_factor**( **gdsl\_stack\_t** *S*, **ulong** *G* )

Set the growing factor of a stack.

Set the growing factor of the stack *S*. This value is the amount of cells to reserve for next insertions. For example, if you set this value to 10, each time the number of elements of *S* reaches 10, then 10 new cells will be reserved for next 10 insertions. It is a way to save time for insertions. To know the actual value of the growing factor, use **gdsl\_stack\_get\_growing\_factor()** (p. 232)

Note

Complexity:  $O(1)$

Precondition

*S* must be a valid **gdsl\_stack\_t**

Parameters

<i>S</i>	The stack to get the growing factor from
<i>G</i>	The new growing factor of <i>S</i> .

Returns

the growing factor of the stack *S*.

See also

**gdsl\_stack\_insert()** (p. 235)

**gdsl\_stack\_get\_growing\_factor()** (p. 232)

#### 4.17.4.12 **gdsl\_element\_t** **gdsl\_stack\_insert**( **gdsl\_stack\_t** *S*, **void \****VALUE* )

Insert an element in a stack (PUSH).

Allocate a new element *E* by calling *S*'s **ALLOC\_F** function on *VALUE*. **ALLOC\_F** is the function pointer passed to **gdsl\_stack\_alloc()** (p. 229). 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**

<b>S</b>	The stack to insert in
<b>VALUE</b>	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. 235)  
**`gdsI_stack_get_growing_factor()`** (p. 232)  
**`gdsI_stack_remove()`** (p. 236)

**Examples:**

**`examples/main_stack.c`**.

**4.17.4.13 `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 `gdsI_stack_t`

**Parameters**

<b>S</b>	The stack to remove the top from
----------	----------------------------------

**Returns**

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

**See also**

**gdsI\_stack\_insert()** (p. 235)

**Examples:**

**examples/main\_stack.c.**

**4.17.4.14** `gdsI_element_t gdsI_stack_search( const gdsI_stack_t S,  
gdsI_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 gdsI\_stack\_t & COMP\_F != NULL

**Parameters**

<i>S</i>	The stack to search the element in
<i>COMP_F</i>	The comparison function used to compare S's element with VALUE
<i>VALUE</i>	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**

**gdsI\_stack\_search\_by\_position()** (p. 237)

**4.17.4.15** `gdsI_element_t gdsI_stack_search_by_position( const gdsI_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. 237)

**Examples:**

**`examples/main_stack.c`**.

**4.17.4.16** `gdsl_element_t` **`gdsl_stack_map_forward`**( `const` `gdsl_stack_t` *S*,  
`gdsl_map_func_t` *MAP\_F*, `void *` *USER\_DATA* )

Parse a stack from bottom to top.

Parse all elements of the stack S from bottom to top. The MAP\_F function is called on each S's element with USER\_DATA argument. If MAP\_F returns GDSL\_MAP\_STOP, then **`gdsl_stack_map_forward()`** (p. 238) stops and returns its last examined element.

**Note**

Complexity:  $O(|S|)$

**Precondition**

S must be a valid `gdsl_stack_t` & *MAP\_F* != NULL

**Parameters**

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



See also

**gdsl\_stack\_map\_backward()** (p. 239)

Examples:

**examples/main\_stack.c.**

4.17.4.17 **gdsl\_element\_t** **gdsl\_stack\_map\_backward**( **const** **gdsl\_stack\_t** *S*,  
**gdsl\_map\_func\_t** *MAP\_F*, **void** \* *USER\_DATA* )

Parse a stack from top to bottom.

Parse all elements of the stack *S* from top to bottom. The *MAP\_F* function is called on each *S*'s element with *USER\_DATA* argument. If *MAP\_F* returns **GDSL\_MAP\_STOP**, then **gdsl\_stack\_map\_backward()** (p. 239) stops and returns its last examined element.

Note

Complexity:  $O(|S|)$

Precondition

*S* must be a valid **gdsl\_stack\_t** & *MAP\_F* != NULL

Parameters

<i>S</i>	The stack to parse
<i>MAP_F</i>	The map function to apply on each <i>S</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\_stack\_map\_forward()** (p. 238)

4.17.4.18 **void** **gdsl\_stack\_write**( **const** **gdsl\_stack\_t** *S*, **gdsl\_write\_func\_t** *WRITE\_F*,  
**FILE** \* *OUTPUT\_FILE*, **void** \* *USER\_DATA* )

Write all the elements of a stack to a file.

Write the elements of the stack *S* to *OUTPUT\_FILE*, using *WRITE\_F* function. -  
Additionnal *USER\_DATA* argument could be passed to *WRITE\_F*.

**Note**

Complexity:  $O(|S|)$

**Precondition**

S must be a valid `gdsi_stack_t` & `OUTPUT_FILE` != NULL & `WRITE_F` != 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_xml()`** (p. 240)

**`gdsi_stack_dump()`** (p. 241)

**4.17.4.19** `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. 239)

**gdsI\_stack\_dump()** (p. 241)

Examples:

**examples/main\_stack.c.**

4.17.4.20 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

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

See also

**gdsI\_stack\_write()** (p. 239)

**gdsI\_stack\_write\_xml()** (p. 240)

Examples:

**examples/main\_stack.c.**

## 4.18 GDSL types

### 4.18.1

#### 4.18.2 Copyright

Copyright (C) 1998-2017 Nicolas Darnis <ndarnis@free.fr>.

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

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

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

### Typedefs

- typedef void \* **gdsl\_element\_t**  
*GDSL element type.*
- typedef **gdsl\_element\_t**(\* **gdsl\_alloc\_func\_t**)(void \*USER\_DATA)  
*GDSL Alloc element function type.*
- typedef void(\* **gdsl\_free\_func\_t**)(**gdsl\_element\_t** E)  
*GDSL Free element function type.*
- typedef **gdsl\_element\_t**(\* **gdsl\_copy\_func\_t**)(const **gdsl\_element\_t** E)  
*GDSL Copy element function type.*
- typedef int(\* **gdsl\_map\_func\_t**)(const **gdsl\_element\_t** E, **gdsl\_location\_t** LOCATION, void \*USER\_DATA)  
*GDSL Map element function type.*
- typedef long int(\* **gdsl\_compare\_func\_t**)(const **gdsl\_element\_t** E, void \*VALUE)  
*GDSL Comparison element function type.*
- typedef void(\* **gdsl\_write\_func\_t**)(const **gdsl\_element\_t** E, FILE \*OUTPUT\_FILE, **gdsl\_location\_t** LOCATION, void \*USER\_DATA)  
*GDSL Write element function type.*
- typedef unsigned long int **ulong**
- typedef unsigned short int **ushort**

### Enumerations

- enum **gdsl\_constant\_t** { **GDSL\_ERR\_MEM\_ALLOC** = -1, **GDSL\_MAP\_STOP** = 0, **GDSL\_MAP\_CONT** = 1, **GDSL\_INSERTED**, **GDSL\_FOUND** }
- GDSL Constants.*

- enum **gdsl\_location\_t** { **GDSL\_LOCATION\_UNDEF** = 0, **GDSL\_LOCATION\_HEAD** = 1, **GDSL\_LOCATION\_ROOT** = 1, **GDSL\_LOCATION\_TOP** = 1, **GDSL\_LOCATION\_TAIL** = 2, **GDSL\_LOCATION\_LEAF** = 2, **GDSL\_LOCATION\_BOTTOM** = 2, **GDSL\_LOCATION\_FIRST** = 1, **GDSL\_LOCATION\_LAST** = 2, **GDSL\_LOCATION\_FIRST\_COL** = 1, **GDSL\_LOCATION\_LAST\_COL** = 2, **GDSL\_LOCATION\_FIRST\_ROW** = 4, **GDSL\_LOCATION\_LAST\_ROW** = 8 }
- enum **bool** { **FALSE** = 0, **TRUE** = 1 }

### 4.18.3 Typedef Documentation

#### 4.18.3.1 typedef void\* **gdsl\_element\_t**

GDSL element type.

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

Definition at line 146 of file `gdsl_types.h`.

#### 4.18.3.2 typedef **gdsl\_element\_t**(\***gdsl\_alloc\_func\_t**)(void\***USER\_DATA**)

GDSL Alloc element function type.

This function type is for allocating a new `gdsl_element_t` variable. The `USER_DATA` argument should be used to fill-in the new element.

##### Parameters

<i>USER_DATA</i>	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. 243)

Definition at line 160 of file `gdsl_types.h`.

#### 4.18.3.3 typedef void(\***gdsl\_free\_func\_t**)(**gdsl\_element\_t**E)

GDSL Free element function type.

This function type is for freeing a `gdsl_element_t` variable. The element must have been previously allocated by a function of `gdsl_alloc_func_t` type. A free function according to `gdsl_free_func_t` must free the resources allocated by the corresponding call to the

function of type `gdsl_alloc_func_t`. The GDSL functions doesn't check if `E != NULL` before calling this function.

#### Parameters

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

#### See also

**`gdsl_alloc_func_t`** (p. 243)

Definition at line 178 of file `gdsl_types.h`.

#### 4.18.3.4 `typedef gdsl_element_t(*gdsl_copy_func_t)(const gdsl_element_t E)`

GDSL Copy element function type.

This function type is for copying `gdsl_element_t` variables.

#### Parameters

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

#### Returns

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

Definition at line 191 of file `gdsl_types.h`.

#### 4.18.3.5 `typedef int(*gdsl_map_func_t)(const gdsl_element_t E, gdsl_location_t LOCATION, void *USER_DATA)`

GDSL Map element function type.

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

#### Parameters

<i>E</i>	The actually mapped <code>gdsl_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 208 of file gdsl\_types.h.

**4.18.3.6** `typedef long int(* gdsl_compare_func_t)(const gdsl_element_t E, void *VALUE)`

GDSDL Comparison element function type.

This function type is used to compare a `gdsl_element_t` variable with a user value. The `E` argument is always the one in the GDSDL data structure, `VALUE` is always the one the user wants to compare `E` with.

#### Parameters

<i>E</i>	The <code>gdsl_element_t</code> variable contained into the data structure to compare from.
<i>VALUE</i>	The user data to compare <code>E</code> 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 229 of file gdsl\_types.h.

**4.18.3.7** `typedef void(* gdsl_write_func_t)(const gdsl_element_t E, FILE *OUTPUT_FILE, gdsl_location_t LOCATION, void *USER_DATA)`

GDSDL Write element function type.

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

#### Parameters

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

Definition at line 245 of file gdsl\_types.h.

**4.18.3.8** `typedef unsigned long int ulong`

Definition at line 258 of file gdsl\_types.h.

#### 4.18.3.9 typedef unsigned short int ushort

Definition at line 262 of file gdsf\_types.h.

### 4.18.4 Enumeration Type Documentation

#### 4.18.4.1 enum gdsf\_constant\_t

GDSF Constants.

Enumerator:

**GDSF\_ERR\_MEM\_ALLOC** Memory allocation error

**GDSF\_MAP\_STOP** For stopping a parsing function

**GDSF\_MAP\_CONT** For continuing a parsing function

**GDSF\_INSERTED** To indicate an inserted value

**GDSF\_FOUND** To indicate a founded value

Definition at line 64 of file gdsf\_types.h.

#### 4.18.4.2 enum gdsf\_location\_t

Enumerator:

**GDSF\_LOCATION\_UNDEF** Element position undefined

**GDSF\_LOCATION\_HEAD** Element is at head position

**GDSF\_LOCATION\_ROOT** Element is on leaf position

**GDSF\_LOCATION\_TOP** Element is at top position

**GDSF\_LOCATION\_TAIL** Element is at tail position

**GDSF\_LOCATION\_LEAF** Element is on root position

**GDSF\_LOCATION\_BOTTOM** Element is at bottom position

**GDSF\_LOCATION\_FIRST** Element is the first

**GDSF\_LOCATION\_LAST** Element is the last

**GDSF\_LOCATION\_FIRST\_COL** Element is on first column

**GDSF\_LOCATION\_LAST\_COL** Element is on last column

**GDSF\_LOCATION\_FIRST\_ROW** Element is on first row

**GDSF\_LOCATION\_LAST\_ROW** Element is on last row

Definition at line 85 of file gdsf\_types.h.



4.18.4.3 enum `bool`

GDSDL boolean type. Defines `_NO_LIBGDSDL_TYPES_` at compilation time if you don't want them.

Enumerator:

***FALSE*** FALSE boolean value

***TRUE*** TRUE boolean value

Definition at line 283 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 **\_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.*

## 5.2 \_gdsI\_bstree.h File Reference

### 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.*

- **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.*

- **\_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.*

- **\_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.*

- **\_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.*

- **\_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.*

- **\_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.*

- **void \_gdsi\_bstree\_write** (const **\_gdsi\_bstree\_t** T, const **\_gdsi\_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 \_gdsi\_bstree\_write\_xml** (const **\_gdsi\_bstree\_t** T, const **\_gdsi\_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 \_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.*

## 5.3 \_gdsi\_list.h File Reference

### Typedefs

- **typedef \_gdsi\_node\_t \_gdsi\_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 **gdsl\_perm\_position\_t** { **GDSL\_PERM\_POSITION\_FIRST** = 1, **GDSL\_PERM\_POSITION\_LAST** = 2 }

*This type is for `gdsl_perm_write_func_t`.*

## Functions

- **gdsl\_perm\_t gdsl\_perm\_alloc** (const char \*NAME, const **ulong** N)  
*Create a new permutation.*
- void **gdsl\_perm\_free** (gdsl\_perm\_t P)  
*Destroy a permutation.*
- **gdsl\_perm\_t gdsl\_perm\_copy** (const gdsl\_perm\_t P)  
*Copy a permutation.*
- const char \* **gdsl\_perm\_get\_name** (const gdsl\_perm\_t P)  
*Get the name of a permutation.*
- **ulong gdsl\_perm\_get\_size** (const gdsl\_perm\_t P)  
*Get the size of a permutation.*
- **ulong gdsl\_perm\_get\_element** (const gdsl\_perm\_t P, const **ulong** INDIX)  
*Get the (INDIX+1)-th element from a permutation.*
- **ulong \* gdsl\_perm\_get\_elements\_array** (const gdsl\_perm\_t P)  
*Get the array elements of a permutation.*
- **ulong gdsl\_perm\_linear\_inversions\_count** (const gdsl\_perm\_t P)  
*Count the inversions number into a linear permutation.*
- **ulong gdsl\_perm\_linear\_cycles\_count** (const gdsl\_perm\_t P)  
*Count the cycles number into a linear permutation.*
- **ulong gdsl\_perm\_canonical\_cycles\_count** (const gdsl\_perm\_t P)  
*Count the cycles number into a canonical permutation.*
- **gdsl\_perm\_t gdsl\_perm\_set\_name** (gdsl\_perm\_t P, const char \*NEW\_NAME)  
*Set the name of a permutation.*
- **gdsl\_perm\_t gdsl\_perm\_linear\_next** (gdsl\_perm\_t P)  
*Get the next permutation from a linear permutation.*
- **gdsl\_perm\_t gdsl\_perm\_linear\_prev** (gdsl\_perm\_t P)  
*Get the previous permutation from a linear permutation.*
- **gdsl\_perm\_t gdsl\_perm\_set\_elements\_array** (gdsl\_perm\_t P, const **ulong** \*ARRAY)  
*Initialize a permutation with an array of values.*
- **gdsl\_perm\_t gdsl\_perm\_multiply** (gdsl\_perm\_t RESULT, const gdsl\_perm\_t ALPHA, const gdsl\_perm\_t BETA)  
*Multiply two permutations.*
- **gdsl\_perm\_t gdsl\_perm\_linear\_to\_canonical** (gdsl\_perm\_t Q, const gdsl\_perm\_t P)  
*Convert a linear permutation to its canonical form.*

- **gdsl\_perm\_t 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



## Chapter 6

# Example Documentation

### 6.1 examples/main\_bstree.c

This is an example of how to use `gdsl_bstree` module.

```
/*
 * This file is part of the Generic Data Structures Library (GDSL).
 * Copyright (C) 1998-2017 Nicolas Darnis <ndarnis@free.fr>.
 *
 * GDSL is free software: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * GDSL is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with GDSL. If not, see <http://www.gnu.org/licenses/>.
 *
 * $RCSfile: main_bstree.c,v $
 * $Revision: 1.19 $
 * $Date: 2015/02/17 12:33:16 $
 */

#include <config.h>

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

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

```

#define N 100

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

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

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

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

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

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

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

                if (gds_l_bstree_delete (t, (void *) name))

```

```
        {
            printf ("String '%s' removed from the tree\n", name);
        }
        else
        {
            printf ("String '%s' not found\n", name);
        }
    }
    break;

case 3:
    gdsl_bstree_flush (t);
    break;

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

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

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

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

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

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

case 9:
```

```

        gds1_bstree_write_xml (t, print_string, stdout, NULL);
        break;

    case 10:
        gds1_bstree_dump (t, print_string, stdout, NULL);
        break;

    case 11:
    {
        int i;
        int rc;
        gds1_perm_t  p = gds1_perm_alloc ("p", N);
        gds1_bstree_t nt = gds1_bstree_alloc ("INTEGERS", alloc_integer,
        free_integer, compare_integers);

        gds1_perm_randomize (p);

        for (i = 0; i < N; i++)
        {
            int n = gds1_perm_get_element (p, i);
            gds1_bstree_insert (nt, &n, &rc);
        }

        printf ("Tree's height: %lu\n", gds1_bstree_get_height (nt));
        gds1_bstree_dump (nt, print_integer, stdout, (void*) "");

        gds1_bstree_free (nt);
        gds1_perm_free (p);
    }
    break;
}
}
while (choice != 0);

gds1_bstree_free (t);

exit (EXIT_SUCCESS);
}

```

## 6.2 examples/main\_hash.c

This file is part of the Generic Data Structures Library (GDSDL). Copyright (C) 1998-2017 Nicolas Darnis <ndarnis@free.fr>.

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

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

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

RCSfile:

gdsl\_hash.h (p. 257),v

Revision:

1.26

Date:

2015/02/17 12:22:56

This is an example of how to use gdsl\_hash module.

```
/*
 * This file is part of the Generic Data Structures Library (GDSL).
 * Copyright (C) 1998-2017 Nicolas Darnis <ndarnis@free.fr>.
 *
 * GDSL is free software: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * GDSL is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with GDSL. If not, see <http://www.gnu.org/licenses/>.
 *
 * $RCSfile: main_hash.c,v $
 * $Revision: 1.25 $
 * $Date: 2015/02/17 12:33:16 $
 */

#include <config.h>

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

#include "gdsl_types.h"
#include "gdsl_hash.h"
#include "_strings.h"

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

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

```

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

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

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

    return (gdsl_element_t) e;
}

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

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

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

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

    mtrace ();

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

    do
    {
        printf ("\t\tMENU - HASH\n\n");
        printf ("\t1> Insert\n");
        printf ("\t2> Search\n");
        printf ("\t3> Remove\n");

```



```
printf ("\t4> Display\n");
printf ("\t5> Flush\n");
printf ("\t6> Fill factor\n");
printf ("\t7> Dump\n");
printf ("\t8> XML display\n");
printf ("\t0> Quit\n\n");
printf ("\t\tYour choice: ");
scanf ("%d", &choice);

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

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

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

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

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

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

case 3:
    {
        char nom[50];
        gdsl_element_t e;

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

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

```

        }
        break;

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

    case 5:
        gdsl_hash_flush (ht);
        break;

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

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

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

gdsl_hash_free (ht);

exit (EXIT_SUCCESS);
}

```

### 6.3 examples/main\_heap.c

This is an example of how to use gdsl\_heap module.

```

/*
 * This file is part of the Generic Data Structures Library (GDSDL).
 * Copyright (C) 1998-2017 Nicolas Darnis <ndarnis@free.fr>.
 *
 * GDSDL is free software: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * GDSDL is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with GDSDL. If not, see <http://www.gnu.org/licenses/>.
 *
 * $RCSfile: main_heap.c,v $
 * $Revision: 1.13 $
 * $Date: 2015/02/17 12:33:16 $
 */

```

```
#include <config.h>

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

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

#include "_integers.h"

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

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

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

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

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

```
case 2:
    if (!gdsl_heap_is_empty (h))
    {
        gds1_heap_delete_top (h);
    }
    else
    {
        printf ("The heap '%s' is empty\n", gds1_heap_get_name (h));
    }
    break;

case 3:
    {
        long int* top;

        if (!gds1_heap_is_empty (h))
        {
            top = (long int*) gds1_heap_get_top (h);
            printf ("Value = %ld\n", *top);
        }
        else
        {
            printf ("The heap '%s' is empty\n", gds1_heap_get_name (h));
        }
    }
    break;

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

        printf ("Enter integer value: ");
        scanf ("%d", &value);
        v = (long int*) gds1_heap_set_top (h, (void*) &value);
        if (v == NULL)
        {
            printf ("value is greather than all other heap ones\n");
        }
        else
        {
            printf ("old value was: %ld\n", *v);
            free_integer (v);
        }
    }
    break;

case 5:
    if (gds1_heap_is_empty (h))
    {
        printf ("The heap '%s' is empty\n", gds1_heap_get_name (h));
    }
    else
    {
        gds1_heap_flush (h);
    }
    break;
/*
case 6:
    {
        int pos;
        long int* value;
```

```

    printf ("Enter an integer value to search: ");
    scanf ("%d", &pos);

    value = (long int*) gdsl_heap_remove (h, &pos);
    if (value == NULL)
    {
        printf ("Not found\n");
    }
    else
    {
        printf ("Value removed %ld\n", *value);
        free_integer (value);
    }
}
break;
*/
case 7:
    if (gdsl_heap_is_empty (h))
    {
        printf ("The heap '%s' is empty\n", gdsl_heap_get_name (h));
    }
    else
    {
        printf ("%s = ( ", gdsl_heap_get_name (h));
        gdsl_heap_map_forward (h, my_display_integer, NULL);
        printf (")\n");
    }
    break;

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

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

gdsl_heap_free (h);

exit (EXIT_SUCCESS);
}

```

## 6.4 examples/main\_interval\_heap.c

This is an example of how to use `gdsl_interval_heap` module.

```

/*
 * This file is part of the Generic Data Structures Library (GDSL).
 * Copyright (C) 1998-2017 Nicolas Darnis <ndarnis@free.fr>.
 *
 * GDSL is free software: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.

```

```

*
* GDSL is distributed in the hope that it will be useful,
* but WITHOUT ANY WARRANTY; without even the implied warranty of
* MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
* GNU General Public License for more details.
*
* You should have received a copy of the GNU General Public License
* along with GDSL. If not, see <http://www.gnu.org/licenses/>.
*
* $RCSfile: main_interval_heap.c,v $
* $Revision: 1.3 $
* $Date: 2015/02/17 12:33:16 $
*/

#include <config.h>

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

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

#include "_integers.h"

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

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

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

```

```
    return value;
}

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

    return value;
}

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

    insert_value(h, 2);
    insert_value(h, 30);
    insert_value(h, 3);
    insert_value(h, 20);
    insert_value(h, 4);
    insert_value(h, 25);
    insert_value(h, 8);
    insert_value(h, 16);
    insert_value(h, 4);
    //exit(0);
    insert_value(h, 10);
    insert_value(h, 10);
    insert_value(h, 15);
    insert_value(h, 5);
    insert_value(h, 12);
    insert_value(h, 8);
    insert_value(h, 16);
    insert_value(h, 9);
    insert_value(h, 15);
    insert_value(h, 5);

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

    remove_min(h);

    remove_max(h);

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

    gdsl_interval_heap_flush(h);

    assert(gdsl_interval_heap_get_size(h) == 0);

    gdsl_interval_heap_free (h);
}

void test2() {
    gdsl_interval_heap_t h = gdsl_interval_heap_alloc ("H", alloc_integer,
```

```

        free_integer, compare_integers);

gdsl_interval_heap_flush(h);

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

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

assert(gdsl_interval_heap_get_size(h) == 0);

gsdl_interval_heap_free (h);
}

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

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

void test3() {
    int i, len=2000;
    long *e;
    gsdl_interval_heap_t h = gsdl_interval_heap_alloc ("H", alloc_integer,
        free_integer, compare_integers);
    gsdl_interval_heap_flush(h);
    long **removed = malloc(len * sizeof(long *));

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

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

        removed[i] = e;
    }

    check_removed(removed, len/2);

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

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

```



```

        e = remove_max(h);

        removed[i] = e;
    }

    check_removed(removed, len/2);

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

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

    gdsi_interval_heap_t h = gdsi_interval_heap_alloc ("H", alloc_integer,
        free_integer, compare_integers);
    gdsi_interval_heap_flush(h);

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

    //assert(gdsi_interval_heap_get_size(h) == 0);
    gdsi_interval_heap_free (h);
}

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

```

## 6.5 examples/main\_list.c

This is an example of how to use gdsi\_list module.

```

/*
 * This file is part of the Generic Data Structures Library (GDSL).
 * Copyright (C) 1998-2017 Nicolas Darnis <ndarnis@free.fr>.
 *
 * GDSL is free software: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.

```

```

*
* GDSL is distributed in the hope that it will be useful,
* but WITHOUT ANY WARRANTY; without even the implied warranty of
* MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
* GNU General Public License for more details.
*
* You should have received a copy of the GNU General Public License
* along with GDSL. If not, see <http://www.gnu.org/licenses/>.
*
* $RCSfile: main_list.c,v $
* $Revision: 1.19 $
* $Date: 2015/02/17 12:33:16 $
*/

#include <config.h>

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

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

#define PERMUTATION_NB 26

static void
affiche_liste_chaines_fwd (gdsl_list_t l)
{
    gdsl_element_t e;
    gdsl_list_cursor_t c = gdsl_list_cursor_alloc (1);

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

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

    printf ("\n");

    gdsl_list_cursor_free (c);
}

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

static void
affiche_liste_chaines_bwd (gdsl_list_t l)
{

```

```

    printf ("%s (<-) = ( ", gds_l_list_get_name (l));

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

    printf ("\n");
}

int main (void)
{
    int choix = 0;

    gds_l_list_t l = gds_l_list_alloc ("MY LIST", alloc_string, free_string);

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

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

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

                do
                {
                    int choix;

                    printf ("\t\tMENU - CELL INSERTION\n\n");
                    printf ("\t1> Insert cell at the beginning of the list\n");
                    printf ("\t2> Insert cell at end of list\n");
                    printf ("\t3> Insert cell after another cell\n");
                    printf ("\t4> Insert cell before another cell\n");
                    printf ("\t5> Display the list\n");
                    printf ("\t0> RETURN TO MAIN MENU\n\n");
                    printf ("\t\tYour choice: ");
                    scanf ("%d", &choix );

                    switch (choix)
                    {
                    case 1:
                        {

```

```

        gdsl_list_insert_head (l, nom);
        done = 1;
    }
    break;

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

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

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

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

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

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

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

```

```
        done = 1;
    }
    gdsl_list_cursor_free (c);
}
break;

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

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

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

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

case 4:
{
    char nom[100];

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

        if (!gdsl_list_delete (l, compare_strings, nom))
```

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

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

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

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

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

case 9:
    if (gdsl_list_is_empty (l))
    {
        printf ("The list is empty.\n");
    }
    else
    {
        gdsl_list_dump (l, print_string, stdout, NULL);
    }
    break;

case 10:
    if (gdsl_list_is_empty (l))
```

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

case 11:
    {
        int pos;
        gdsl_element_t e;

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

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

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

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

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

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

case 14:
    if (gdsl_list_is_empty (l))
    {
        printf ("The list is empty.\n");
    }
    else
    {
        printf ("Max Element: %s\n", (char*) gdsl_list_search_max (l,
compare_strings));
    }
    break;
```

```

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

        gdsl_perm_randomize (p);

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

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

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

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

            gdsl_list_cursor_insert_before (c, toto);

            gdsl_list_cursor_step_backward (c);
            gdsl_list_cursor_delete_after (c);
        }

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

gdsl_list_free (l);

exit (EXIT_SUCCESS);
}

```



## 6.6 examples/main\_llbintree.c

This is an example of how to use `_gdsl_bintree` module.

```

/*
 * This file is part of the Generic Data Structures Library (GDSL).
 * Copyright (C) 1998-2017 Nicolas Darnis <ndarnis@free.fr>.
 *
 * GDSL is free software: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * GDSL is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with GDSL. If not, see <http://www.gnu.org/licenses/>.
 *
 * $RCSfile: main_llbintree.c,v $
 * $Revision: 1.14 $
 * $Date: 2015/02/17 12:33:16 $
 */

#include <config.h>

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

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

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

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

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

```

```

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

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

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

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

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

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

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

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

    exit (EXIT_SUCCESS);
}

```

## 6.7 examples/main\_lbstree.c

This is an example of how to use `_gdsl_bstree` module.

```

/*
 * This file is part of the Generic Data Structures Library (GDSL).
 * Copyright (C) 1998-2017 Nicolas Darnis <ndarnis@free.fr>.
 *
 * GDSL is free software: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.

```

```

*
* GDSL is distributed in the hope that it will be useful,
* but WITHOUT ANY WARRANTY; without even the implied warranty of
* MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
* GNU General Public License for more details.
*
* You should have received a copy of the GNU General Public License
* along with GDSL. If not, see <http://www.gnu.org/licenses/>.
*
* $RCSfile: main_llbstree.c,v $
* $Revision: 1.13 $
* $Date: 2015/02/17 12:33:16 $
*/

#include <config.h>

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

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

#define N 100

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

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

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

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

```

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

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

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

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

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

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

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

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

    printf ("T:\n");

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

    {
        int i;
```

```

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

gdsl_perm_randomize (p);

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

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

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

exit (EXIT_SUCCESS);
}

```

## 6.8 examples/main\_llist.c

This is an example of how to use `_gdsl_list` module.

```

/*
 * This file is part of the Generic Data Structures Library (GDSL).
 * Copyright (C) 1998-2017 Nicolas Darnis <ndarnis@free.fr>.
 *
 * GDSL is free software: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * GDSL is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with GDSL. If not, see <http://www.gnu.org/licenses/>.
 *
 * $RCSfile: main_llist.c,v $
 * $Revision: 1.13 $
 * $Date: 2015/02/17 12:33:16 $
 */

#include <config.h>

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

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

```

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

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

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

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

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

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

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

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

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

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

    _gdsl_list_free (a, free_string);

    exit (EXIT_SUCCESS);
}
```

## 6.9 examples/main\_perm.c

This is an example of how to use gds\_l\_perm module.

```
/*
 * This file is part of the Generic Data Structures Library (GDSL).
 * Copyright (C) 1998-2017 Nicolas Darnis <ndarnis@free.fr>.
 *
 * GDSL is free software: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * GDSL is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with GDSL. If not, see <http://www.gnu.org/licenses/>.
 *
 * $RCSfile: main_perm.c,v $
 * $Revision: 1.19 $
 * $Date: 2015/02/17 12:33:16 $
 */

#include <config.h>

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

#include "gds_l_types.h"
#include "gds_l_perm.h"
#include "_integers.h"

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

static void
write (const gds_l_element_t e, FILE* file, gds_l_location_t pos, void* user_data)
{
    unsigned n = * (unsigned*) e;

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

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

```

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

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

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

    n = atoi (argv[1]);

    c_alpha = gdsl_perm_alloc ("c_alpha", n);

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

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

    gdsl_perm_reverse (l_alpha);

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

    gdsl_perm_reverse (l_alpha);
    gdsl_perm_inverse (l_alpha);

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

    gdsl_perm_inverse (l_alpha);

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

    gdsl_perm_linear_to_canonical (c_alpha, l_alpha);
    printf ("cycles(alpha) = ");

```



```

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

gdsl_perm_canonical_to_linear (l_alpha, c_alpha);

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

gdsl_perm_free (l_alpha);
gdsl_perm_free (c_alpha);

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

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

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

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

    exit (EXIT_SUCCESS);
}

```

## 6.10 examples/main\_queue.c

This is an example of how to use `gdsl_queue` module.

```

/*
 * This file is part of the Generic Data Structures Library (GDSL).
 * Copyright (C) 1998-2017 Nicolas Darnis <ndarnis@free.fr>.
 *
 * GDSL is free software: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 */

```

```
* GDSL is distributed in the hope that it will be useful,
* but WITHOUT ANY WARRANTY; without even the implied warranty of
* MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
* GNU General Public License for more details.
*
* You should have received a copy of the GNU General Public License
* along with GDSL. If not, see <http://www.gnu.org/licenses/>.
*
* $RCSfile: main_queue.c,v $
* $Revision: 1.13 $
* $Date: 2015/02/17 12:33:16 $
*/

#include <config.h>

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

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

#include "_integers.h"

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

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

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

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

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

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

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

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

    case 3:
        {
            if (!gdsl_queue_is_empty (q))
            {
                int head = *(int*) gdsl_queue_get_head (q);
                printf ("Head = %d\n", head);
            }
            else
            {
                printf ("The queue '%s' is empty\n", gdsl_queue_get_name (q));
            }
        }
        break;

    case 4:
        {
            if (!gdsl_queue_is_empty (q))
            {
                int tail = *(int*) gdsl_queue_get_tail (q);
                printf ("Tail = %d\n", tail);
            }
            else
        }
```

```
        {
            printf ("The queue '%s' is empty\n", gds1_queue_get_name (q));
        }
    }
    break;

case 5:
    if (gds1_queue_is_empty (q))
    {
        printf ("The queue '%s' is empty\n", gds1_queue_get_name (q));
    }
    else
    {
        gds1_queue_flush (q);
    }
    break;

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

        value = (int*) gds1_queue_search_by_position (q, pos);
        if (value != NULL)
        {
            printf ("Value found at position %d = %d\n", pos, *value);
        }
    }
    break;

case 7:
    if (gds1_queue_is_empty (q))
    {
        printf ("The queue '%s' is empty\n", gds1_queue_get_name (q));
    }
    else
    {
        printf ("%s = ", gds1_queue_get_name (q));
        gds1_queue_map_forward (q, my_display_integer, NULL);
    }
    break;

case 8:
    gds1_queue_dump (q, my_write_integer, stdout, NULL);
    break;

case 9:
    gds1_queue_write_xml (q, my_write_integer, stdout, NULL);
    break;
    }
}
while (choice != 0);

gds1_queue_free (q);

exit (EXIT_SUCCESS);
}
```

## 6.11 examples/main\_rbtrees.c

This is an example of how to use gdslib\_rbtrees module.

```

/*
 * This file is part of the Generic Data Structures Library (GDSL).
 * Copyright (C) 1998-2017 Nicolas Darnis <ndarnis@free.fr>.
 *
 * GDSL is free software: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * GDSL is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with GDSL. If not, see <http://www.gnu.org/licenses/>.
 *
 * $RCSfile: main_rbtrees.c,v $
 * $Revision: 1.20 $
 * $Date: 2015/02/17 12:33:16 $
 */

#include <config.h>

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

#include "gdslib_perm.h"
#include "gdslib_types.h"
#include "gdslib_rbtrees.h"
#include "_strings.h"
#include "_integers.h"

#define N 100

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

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

```

```

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

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

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

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

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

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

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

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

```

```
    }
    break;

case 3:
    gdsl_rbtrees_flush (t);
    break;

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

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

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

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

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

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

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

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

```

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

        gdsl_perm_randomize (p);

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

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

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

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

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

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

gdsl_rbtrees_free (t);

exit (EXIT_SUCCESS);
}

```

## 6.12 examples/main\_sort.c

This is an example of how to use `gdsl_sort` module.

```

/*
 * This file is part of the Generic Data Structures Library (GDSL).
 * Copyright (C) 1998-2017 Nicolas Darnis <ndarnis@free.fr>.
 *
 * GDSL is free software: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *

```



```
* GDSL is distributed in the hope that it will be useful,
* but WITHOUT ANY WARRANTY; without even the implied warranty of
* MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
* GNU General Public License for more details.
*
* You should have received a copy of the GNU General Public License
* along with GDSL. If not, see <http://www.gnu.org/licenses/>.
*
* $RCSfile: main_sort.c,v $
* $Revision: 1.2 $
* $Date: 2015/02/17 12:33:17 $
*/

#include <config.h>

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

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

#define N 100
#define M 26

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

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

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

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

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

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

```
    }  
    printf ("\n");  
  
    exit (EXIT_SUCCESS);  
}
```

## 6.13 examples/main\_stack.c

This is an example of how to use gdsl\_stack module.

```
/*  
 * This file is part of the Generic Data Structures Library (GDSL).  
 * Copyright (C) 1998-2017 Nicolas Darnis <ndarnis@free.fr>.  
 *  
 * GDSL is free software: you can redistribute it and/or modify  
 * it under the terms of the GNU General Public License as published by  
 * the Free Software Foundation, either version 3 of the License, or  
 * (at your option) any later version.  
 *  
 * GDSL is distributed in the hope that it will be useful,  
 * but WITHOUT ANY WARRANTY; without even the implied warranty of  
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the  
 * GNU General Public License for more details.  
 *  
 * You should have received a copy of the GNU General Public License  
 * along with GDSL. If not, see <http://www.gnu.org/licenses/>.  
 *  
 * $RCSfile: main_stack.c,v $  
 * $Revision: 1.14 $  
 * $Date: 2015/02/17 12:33:17 $  
 */  
  
#include <config.h>  
  
#include <stdio.h>  
#include <stdlib.h>  
#include <string.h>  
  
#include "gdsl_types.h"  
#include "gdsl_stack.h"  
  
#include "_integers.h"  
  
static int  
my_display_integer (gdsl_element_t e, gdsl_location_t location, void *  
                    user_infos)  
{  
    int* f = (int*) e;  
    printf ("%d ", *f);  
    return GDSL_MAP_CONT;  
}
```

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

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

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

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

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

            case 3:
            {
                int* top;

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

            case 4:
                if (gdsl_stack_is_empty (s))
                {
```

```
        printf ("The stack '%s' is empty\n", gdsl_stack_get_name (s));
    }
    else
    {
        gdsl_stack_flush (s);
    }
    break;

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

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

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

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

case 8:
gdsl_stack_write_xml (s, print_integer, stdout, NULL);
break;
}
}
while (choix != 0);

gdsl_stack_free (s);

exit (EXIT_SUCCESS);
}
```

# Index

## Binary search tree manipulation module,

80

- gdsl\_bstree\_alloc, 81
- gdsl\_bstree\_delete, 88
- gdsl\_bstree\_dump, 94
- gdsl\_bstree\_flush, 83
- gdsl\_bstree\_free, 82
- gdsl\_bstree\_get\_height, 86
- gdsl\_bstree\_get\_name, 84
- gdsl\_bstree\_get\_root, 85
- gdsl\_bstree\_get\_size, 85
- gdsl\_bstree\_insert, 87
- gdsl\_bstree\_is\_empty, 84
- gdsl\_bstree\_map\_infix, 91
- gdsl\_bstree\_map\_postfix, 91
- gdsl\_bstree\_map\_prefix, 90
- gdsl\_bstree\_remove, 88
- gdsl\_bstree\_search, 89
- gdsl\_bstree\_set\_name, 86
- gdsl\_bstree\_t, 81
- gdsl\_bstree\_write, 92
- gdsl\_bstree\_write\_xml, 93

## Doubly-linked list manipulation module,

139

- gdsl\_list\_alloc, 142
- gdsl\_list\_cursor\_alloc, 161
- gdsl\_list\_cursor\_delete, 172
- gdsl\_list\_cursor\_delete\_after, 173
- gdsl\_list\_cursor\_delete\_before, 173
- gdsl\_list\_cursor\_free, 161
- gdsl\_list\_cursor\_get\_content, 168
- gdsl\_list\_cursor\_has\_pred, 167
- gdsl\_list\_cursor\_has\_succ, 166
- gdsl\_list\_cursor\_insert\_after, 169
- gdsl\_list\_cursor\_insert\_before, 169
- gdsl\_list\_cursor\_is\_on\_head, 165
- gdsl\_list\_cursor\_is\_on\_tail, 166
- gdsl\_list\_cursor\_move\_to\_head, 162
- gdsl\_list\_cursor\_move\_to\_position, 164
- gdsl\_list\_cursor\_move\_to\_tail, 162

- gdsl\_list\_cursor\_move\_to\_value, 163

- gdsl\_list\_cursor\_remove, 170
- gdsl\_list\_cursor\_remove\_after, 171
- gdsl\_list\_cursor\_remove\_before, 172
- gdsl\_list\_cursor\_set\_content, 168
- gdsl\_list\_cursor\_step\_backward, 165
- gdsl\_list\_cursor\_step\_forward, 164
- gdsl\_list\_cursor\_t, 142
- gdsl\_list\_delete, 153
- gdsl\_list\_delete\_head, 151
- gdsl\_list\_delete\_tail, 152
- gdsl\_list\_dump, 160
- gdsl\_list\_flush, 144
- gdsl\_list\_free, 143
- gdsl\_list\_get\_head, 146
- gdsl\_list\_get\_name, 144
- gdsl\_list\_get\_size, 145
- gdsl\_list\_get\_tail, 146
- gdsl\_list\_insert\_head, 148
- gdsl\_list\_insert\_tail, 148
- gdsl\_list\_is\_empty, 145
- gdsl\_list\_map\_backward, 158
- gdsl\_list\_map\_forward, 157
- gdsl\_list\_remove, 151
- gdsl\_list\_remove\_head, 149
- gdsl\_list\_remove\_tail, 150
- gdsl\_list\_search, 153
- gdsl\_list\_search\_by\_position, 154
- gdsl\_list\_search\_max, 155
- gdsl\_list\_search\_min, 156
- gdsl\_list\_set\_name, 147
- gdsl\_list\_sort, 156
- gdsl\_list\_t, 142
- gdsl\_list\_write, 158
- gdsl\_list\_write\_xml, 159

## FALSE

- GDSL types, 247

## GDSL types, 242

- FALSE, 247

- GDSL\_ERR\_MEM\_ALLOC, 246

- GDSL\_FOUND, 246
- GDSL\_INSERTED, 246
- GDSL\_LOCATION\_BOTTOM, 246
- GDSL\_LOCATION\_FIRST, 246
- GDSL\_LOCATION\_FIRST\_COL, 246
- GDSL\_LOCATION\_FIRST\_ROW, 246
- GDSL\_LOCATION\_HEAD, 246
- GDSL\_LOCATION\_LAST, 246
- GDSL\_LOCATION\_LAST\_COL, 246
- GDSL\_LOCATION\_LAST\_ROW, 246
- GDSL\_LOCATION\_LEAF, 246
- GDSL\_LOCATION\_ROOT, 246
- GDSL\_LOCATION\_TAIL, 246
- GDSL\_LOCATION\_TOP, 246
- GDSL\_LOCATION\_UNDEF, 246
- GDSL\_MAP\_CONT, 246
- GDSL\_MAP\_STOP, 246
- TRUE, 247
- bool, 246
- gdsl\_alloc\_func\_t, 243
- gdsl\_compare\_func\_t, 245
- gdsl\_constant\_t, 246
- gdsl\_copy\_func\_t, 244
- gdsl\_element\_t, 243
- gdsl\_free\_func\_t, 243
- gdsl\_location\_t, 246
- gdsl\_map\_func\_t, 244
- gdsl\_write\_func\_t, 245
- ulong, 245
- ushort, 245
- GDSL\_ERR\_MEM\_ALLOC
  - GDSL types, 246
- GDSL\_FOUND
  - GDSL types, 246
- GDSL\_INSERTED
  - GDSL types, 246
- GDSL\_LOCATION\_BOTTOM
  - GDSL types, 246
- GDSL\_LOCATION\_FIRST
  - GDSL types, 246
- GDSL\_LOCATION\_FIRST\_COL
  - GDSL types, 246
- GDSL\_LOCATION\_FIRST\_ROW
  - GDSL types, 246
- GDSL\_LOCATION\_HEAD
  - GDSL types, 246
- GDSL\_LOCATION\_LAST
  - GDSL types, 246
- GDSL\_LOCATION\_LAST\_COL
  - GDSL types, 246
- GDSL\_LOCATION\_LAST\_ROW
  - GDSL types, 246
- GDSL\_LOCATION\_LEAF
  - GDSL types, 246
- GDSL\_LOCATION\_ROOT
  - GDSL types, 246
- GDSL\_LOCATION\_TAIL
  - GDSL types, 246
- GDSL\_LOCATION\_TOP
  - GDSL types, 246
- GDSL\_LOCATION\_UNDEF
  - GDSL types, 246
- GDSL\_MAP\_CONT
  - GDSL types, 246
- GDSL\_MAP\_STOP
  - GDSL types, 246
- GDSL\_MAX
  - Various macros module, 175
- GDSL\_MIN
  - Various macros module, 176
- GDSL\_PERM\_POSITION\_FIRST
  - Permutation manipulation module, 179
- GDSL\_PERM\_POSITION\_LAST
  - Permutation manipulation module, 179
- Hashtable manipulation module, 95
  - gdsl\_hash, 97
  - gdsl\_hash\_alloc, 98
  - gdsl\_hash\_delete, 106
  - gdsl\_hash\_dump, 110
  - gdsl\_hash\_flush, 99
  - gdsl\_hash\_free, 99
  - gdsl\_hash\_func\_t, 97
  - gdsl\_hash\_get\_entries\_number, 101
  - gdsl\_hash\_get\_fill\_factor, 103
  - gdsl\_hash\_get\_lists\_max\_size, 101
  - gdsl\_hash\_get\_longest\_list\_size, 102
  - gdsl\_hash\_get\_name, 100
  - gdsl\_hash\_get\_size, 102
  - gdsl\_hash\_insert, 104
  - gdsl\_hash\_map, 108
  - gdsl\_hash\_modify, 107
  - gdsl\_hash\_remove, 105
  - gdsl\_hash\_search, 107
  - gdsl\_hash\_set\_name, 104

- gdsl\_hash\_t, 96
- gdsl\_hash\_write, 109
- gdsl\_hash\_write\_xml, 110
- gdsl\_key\_func\_t, 96
- Heap manipulation module, 112
  - gdsl\_heap\_alloc, 113
  - gdsl\_heap\_delete\_top, 120
  - gdsl\_heap\_dump, 123
  - gdsl\_heap\_flush, 115
  - gdsl\_heap\_free, 114
  - gdsl\_heap\_get\_name, 115
  - gdsl\_heap\_get\_size, 116
  - gdsl\_heap\_get\_top, 116
  - gdsl\_heap\_insert, 119
  - gdsl\_heap\_is\_empty, 117
  - gdsl\_heap\_map\_forward, 121
  - gdsl\_heap\_remove\_top, 120
  - gdsl\_heap\_set\_name, 118
  - gdsl\_heap\_set\_top, 118
  - gdsl\_heap\_t, 113
  - gdsl\_heap\_write, 122
  - gdsl\_heap\_write\_xml, 122
- Interval Heap manipulation module, 125
  - gdsl\_interval\_heap\_alloc, 127
  - gdsl\_interval\_heap\_delete\_max, 135
  - gdsl\_interval\_heap\_delete\_min, 134
  - gdsl\_interval\_heap\_dump, 138
  - gdsl\_interval\_heap\_flush, 128
  - gdsl\_interval\_heap\_free, 127
  - gdsl\_interval\_heap\_get\_max, 134
  - gdsl\_interval\_heap\_get\_min, 134
  - gdsl\_interval\_heap\_get\_name, 129
  - gdsl\_interval\_heap\_get\_size, 129
  - gdsl\_interval\_heap\_insert, 131
  - gdsl\_interval\_heap\_is\_empty, 130
  - gdsl\_interval\_heap\_map\_forward, 136
  - gdsl\_interval\_heap\_remove\_max, 132
  - gdsl\_interval\_heap\_remove\_min, 133
  - gdsl\_interval\_heap\_set\_max\_size, 130
  - gdsl\_interval\_heap\_set\_name, 131
  - gdsl\_interval\_heap\_t, 126
  - gdsl\_interval\_heap\_write, 136
  - gdsl\_interval\_heap\_write\_xml, 137
- Low level binary tree manipulation module, 9
  - \_gdsl\_bintree\_alloc, 12
  - \_gdsl\_bintree\_copy, 13
  - \_gdsl\_bintree\_dump, 29
  - \_gdsl\_bintree\_free, 13
  - \_gdsl\_bintree\_get\_content, 16
  - \_gdsl\_bintree\_get\_height, 19
  - \_gdsl\_bintree\_get\_left, 17
  - \_gdsl\_bintree\_get\_left\_ref, 18
  - \_gdsl\_bintree\_get\_parent, 16
  - \_gdsl\_bintree\_get\_right, 18
  - \_gdsl\_bintree\_get\_right\_ref, 19
  - \_gdsl\_bintree\_get\_size, 20
  - \_gdsl\_bintree\_is\_empty, 14
  - \_gdsl\_bintree\_is\_leaf, 15
  - \_gdsl\_bintree\_is\_root, 15
  - \_gdsl\_bintree\_map\_func\_t, 11
  - \_gdsl\_bintree\_map\_infix, 26
  - \_gdsl\_bintree\_map\_postfix, 27
  - \_gdsl\_bintree\_map\_prefix, 25
  - \_gdsl\_bintree\_rotate\_left, 23
  - \_gdsl\_bintree\_rotate\_left\_right, 24
  - \_gdsl\_bintree\_rotate\_right, 23
  - \_gdsl\_bintree\_rotate\_right\_left, 25
  - \_gdsl\_bintree\_set\_content, 20
  - \_gdsl\_bintree\_set\_left, 21
  - \_gdsl\_bintree\_set\_parent, 21
  - \_gdsl\_bintree\_set\_right, 22
  - \_gdsl\_bintree\_t, 11
  - \_gdsl\_bintree\_write, 28
  - \_gdsl\_bintree\_write\_func\_t, 12
  - \_gdsl\_bintree\_write\_xml, 28
- Low-level binary search tree manipulation module, 31
  - \_gdsl\_bstree\_alloc, 33
  - \_gdsl\_bstree\_copy, 35
  - \_gdsl\_bstree\_dump, 47
  - \_gdsl\_bstree\_free, 34
  - \_gdsl\_bstree\_get\_content, 36
  - \_gdsl\_bstree\_get\_height, 40
  - \_gdsl\_bstree\_get\_left, 38
  - \_gdsl\_bstree\_get\_parent, 38
  - \_gdsl\_bstree\_get\_right, 39
  - \_gdsl\_bstree\_get\_size, 39
  - \_gdsl\_bstree\_insert, 40
  - \_gdsl\_bstree\_is\_empty, 35
  - \_gdsl\_bstree\_is\_leaf, 36
  - \_gdsl\_bstree\_is\_root, 37
  - \_gdsl\_bstree\_map\_func\_t, 33
  - \_gdsl\_bstree\_map\_infix, 44
  - \_gdsl\_bstree\_map\_postfix, 45
  - \_gdsl\_bstree\_map\_prefix, 43

- `_gdsl_bstree_remove`, 41
- `_gdsl_bstree_search`, 42
- `_gdsl_bstree_search_next`, 43
- `_gdsl_bstree_t`, 33
- `_gdsl_bstree_write`, 45
- `_gdsl_bstree_write_func_t`, 33
- `_gdsl_bstree_write_xml`, 46
- Low-level doubly-linked list manipulation
  - module, 48
  - `_gdsl_list_alloc`, 49
  - `_gdsl_list_dump`, 57
  - `_gdsl_list_free`, 50
  - `_gdsl_list_get_size`, 51
  - `_gdsl_list_insert_after`, 52
  - `_gdsl_list_insert_before`, 52
  - `_gdsl_list_is_empty`, 50
  - `_gdsl_list_link`, 51
  - `_gdsl_list_map_backward`, 55
  - `_gdsl_list_map_forward`, 54
  - `_gdsl_list_remove`, 53
  - `_gdsl_list_search`, 53
  - `_gdsl_list_t`, 49
  - `_gdsl_list_write`, 55
  - `_gdsl_list_write_xml`, 56
- Low-level doubly-linked node manipulation
  - module, 59
  - `_gdsl_node_alloc`, 61
  - `_gdsl_node_dump`, 67
  - `_gdsl_node_free`, 61
  - `_gdsl_node_get_content`, 63
  - `_gdsl_node_get_pred`, 62
  - `_gdsl_node_get_succ`, 62
  - `_gdsl_node_link`, 65
  - `_gdsl_node_map_func_t`, 60
  - `_gdsl_node_set_content`, 65
  - `_gdsl_node_set_pred`, 64
  - `_gdsl_node_set_succ`, 64
  - `_gdsl_node_t`, 60
  - `_gdsl_node_unlink`, 66
  - `_gdsl_node_write`, 66
  - `_gdsl_node_write_func_t`, 60
  - `_gdsl_node_write_xml`, 67
- Main module, 69
  - `gdsl_get_version`, 69
- Permutation manipulation module, 177
  - `GDSL_PERM_POSITION_FIRST`, 179
  - `GDSL_PERM_POSITION_LAST`, 179
  - `gdsl_perm_alloc`, 180
  - `gdsl_perm_apply_on_array`, 192
  - `gdsl_perm_canonical_cycles_count`, 185
  - `gdsl_perm_canonical_to_linear`, 189
  - `gdsl_perm_copy`, 181
  - `gdsl_perm_data_t`, 179
  - `gdsl_perm_dump`, 194
  - `gdsl_perm_free`, 180
  - `gdsl_perm_get_element`, 183
  - `gdsl_perm_get_elements_array`, 183
  - `gdsl_perm_get_name`, 181
  - `gdsl_perm_get_size`, 182
  - `gdsl_perm_inverse`, 190
  - `gdsl_perm_linear_cycles_count`, 184
  - `gdsl_perm_linear_inversions_count`, 184
  - `gdsl_perm_linear_next`, 186
  - `gdsl_perm_linear_prev`, 187
  - `gdsl_perm_linear_to_canonical`, 189
  - `gdsl_perm_multiply`, 188
  - `gdsl_perm_position_t`, 179
  - `gdsl_perm_randomize`, 191
  - `gdsl_perm_reverse`, 191
  - `gdsl_perm_set_elements_array`, 187
  - `gdsl_perm_set_name`, 186
  - `gdsl_perm_t`, 179
  - `gdsl_perm_write`, 192
  - `gdsl_perm_write_func_t`, 179
  - `gdsl_perm_write_xml`, 193
- Queue manipulation module, 195
  - `gdsl_queue_alloc`, 196
  - `gdsl_queue_dump`, 207
  - `gdsl_queue_flush`, 198
  - `gdsl_queue_free`, 197
  - `gdsl_queue_get_head`, 200
  - `gdsl_queue_get_name`, 198
  - `gdsl_queue_get_size`, 199
  - `gdsl_queue_get_tail`, 201
  - `gdsl_queue_insert`, 202
  - `gdsl_queue_is_empty`, 199
  - `gdsl_queue_map_backward`, 205
  - `gdsl_queue_map_forward`, 205
  - `gdsl_queue_remove`, 203
  - `gdsl_queue_search`, 203
  - `gdsl_queue_search_by_position`, 204
  - `gdsl_queue_set_name`, 201
  - `gdsl_queue_t`, 196
  - `gdsl_queue_write`, 206
  - `gdsl_queue_write_xml`, 207



- Red-black tree manipulation module, 209
  - gdsl\_rbtrees\_alloc, 210
  - gdsl\_rbtrees\_delete, 217
  - gdsl\_rbtrees\_dump, 223
  - gdsl\_rbtrees\_flush, 212
  - gdsl\_rbtrees\_free, 211
  - gdsl\_rbtrees\_get\_name, 212
  - gdsl\_rbtrees\_get\_root, 214
  - gdsl\_rbtrees\_get\_size, 214
  - gdsl\_rbtrees\_height, 215
  - gdsl\_rbtrees\_insert, 216
  - gdsl\_rbtrees\_is\_empty, 213
  - gdsl\_rbtrees\_map\_infix, 220
  - gdsl\_rbtrees\_map\_postfix, 221
  - gdsl\_rbtrees\_map\_prefix, 219
  - gdsl\_rbtrees\_remove, 217
  - gdsl\_rbtrees\_search, 218
  - gdsl\_rbtrees\_set\_name, 215
  - gdsl\_rbtrees\_t, 210
  - gdsl\_rbtrees\_write, 221
  - gdsl\_rbtrees\_write\_xml, 222
- Sort module, 225
  - gdsl\_sort, 225
- Stack manipulation module, 227
  - gdsl\_stack\_alloc, 229
  - gdsl\_stack\_dump, 241
  - gdsl\_stack\_flush, 230
  - gdsl\_stack\_free, 229
  - gdsl\_stack\_get\_bottom, 233
  - gdsl\_stack\_get\_growing\_factor, 232
  - gdsl\_stack\_get\_name, 230
  - gdsl\_stack\_get\_size, 231
  - gdsl\_stack\_get\_top, 233
  - gdsl\_stack\_insert, 235
  - gdsl\_stack\_is\_empty, 232
  - gdsl\_stack\_map\_backward, 239
  - gdsl\_stack\_map\_forward, 238
  - gdsl\_stack\_remove, 236
  - gdsl\_stack\_search, 237
  - gdsl\_stack\_search\_by\_position, 237
  - gdsl\_stack\_set\_growing\_factor, 235
  - gdsl\_stack\_set\_name, 234
  - gdsl\_stack\_t, 228
  - gdsl\_stack\_write, 239
  - gdsl\_stack\_write\_xml, 240
- TRUE
  - GDSL types, 247
- Various macros module, 175
  - GDSL\_MAX, 175
  - GDSL\_MIN, 176
- \_gdsl\_bintree.h, 249
- \_gdsl\_bintree\_alloc
  - Low level binary tree manipulation module, 12
- \_gdsl\_bintree\_copy
  - Low level binary tree manipulation module, 13
- \_gdsl\_bintree\_dump
  - Low level binary tree manipulation module, 29
- \_gdsl\_bintree\_free
  - Low level binary tree manipulation module, 13
- \_gdsl\_bintree\_get\_content
  - Low level binary tree manipulation module, 16
- \_gdsl\_bintree\_get\_height
  - Low level binary tree manipulation module, 19
- \_gdsl\_bintree\_get\_left
  - Low level binary tree manipulation module, 17
- \_gdsl\_bintree\_get\_left\_ref
  - Low level binary tree manipulation module, 18
- \_gdsl\_bintree\_get\_parent
  - Low level binary tree manipulation module, 16
- \_gdsl\_bintree\_get\_right
  - Low level binary tree manipulation module, 18
- \_gdsl\_bintree\_get\_right\_ref
  - Low level binary tree manipulation module, 19
- \_gdsl\_bintree\_get\_size
  - Low level binary tree manipulation module, 20
- \_gdsl\_bintree\_is\_empty
  - Low level binary tree manipulation module, 14
- \_gdsl\_bintree\_is\_leaf
  - Low level binary tree manipulation module, 15
- \_gdsl\_bintree\_is\_root
  - Low level binary tree manipulation module, 15
- \_gdsl\_bintree\_map\_func\_t
  - Low level binary tree manipulation module, 11
- \_gdsl\_bintree\_map\_infix

- Low level binary tree manipulation module, 26
- `_gdsl_bintree_map_postfix`
  - Low level binary tree manipulation module, 27
- `_gdsl_bintree_map_prefix`
  - Low level binary tree manipulation module, 25
- `_gdsl_bintree_rotate_left`
  - Low level binary tree manipulation module, 23
- `_gdsl_bintree_rotate_left_right`
  - Low level binary tree manipulation module, 24
- `_gdsl_bintree_rotate_right`
  - Low level binary tree manipulation module, 23
- `_gdsl_bintree_rotate_right_left`
  - Low level binary tree manipulation module, 25
- `_gdsl_bintree_set_content`
  - Low level binary tree manipulation module, 20
- `_gdsl_bintree_set_left`
  - Low level binary tree manipulation module, 21
- `_gdsl_bintree_set_parent`
  - Low level binary tree manipulation module, 21
- `_gdsl_bintree_set_right`
  - Low level binary tree manipulation module, 22
- `_gdsl_bintree_t`
  - Low level binary tree manipulation module, 11
- `_gdsl_bintree_write`
  - Low level binary tree manipulation module, 28
- `_gdsl_bintree_write_func_t`
  - Low level binary tree manipulation module, 12
- `_gdsl_bintree_write_xml`
  - Low level binary tree manipulation module, 28
- `_gdsl_bstree.h`, 251
- `_gdsl_bstree_alloc`
  - Low-level binary search tree manipulation module, 33
- `_gdsl_bstree_copy`
  - Low-level binary search tree manipulation module, 35
- `_gdsl_bstree_dump`
  - Low-level binary search tree manipulation module, 47
- `_gdsl_bstree_free`
  - Low-level binary search tree manipulation module, 34
- `_gdsl_bstree_get_content`
  - Low-level binary search tree manipulation module, 36
- `_gdsl_bstree_get_height`
  - Low-level binary search tree manipulation module, 40
- `_gdsl_bstree_get_left`
  - Low-level binary search tree manipulation module, 38
- `_gdsl_bstree_get_parent`
  - Low-level binary search tree manipulation module, 38
- `_gdsl_bstree_get_right`
  - Low-level binary search tree manipulation module, 39
- `_gdsl_bstree_get_size`
  - Low-level binary search tree manipulation module, 39
- `_gdsl_bstree_insert`
  - Low-level binary search tree manipulation module, 40
- `_gdsl_bstree_is_empty`
  - Low-level binary search tree manipulation module, 35
- `_gdsl_bstree_is_leaf`
  - Low-level binary search tree manipulation module, 36
- `_gdsl_bstree_is_root`
  - Low-level binary search tree manipulation module, 37
- `_gdsl_bstree_map_func_t`
  - Low-level binary search tree manipulation module, 33
- `_gdsl_bstree_map_infix`
  - Low-level binary search tree manipulation module, 44
- `_gdsl_bstree_map_postfix`
  - Low-level binary search tree manipulation module, 45
- `_gdsl_bstree_map_prefix`
  - Low-level binary search tree manipulation module, 43

- `_gdsl_bstree_remove`  
Low-level binary search tree manipulation module, 41
  - `_gdsl_bstree_search`  
Low-level binary search tree manipulation module, 42
  - `_gdsl_bstree_search_next`  
Low-level binary search tree manipulation module, 43
  - `_gdsl_bstree_t`  
Low-level binary search tree manipulation module, 33
  - `_gdsl_bstree_write`  
Low-level binary search tree manipulation module, 45
  - `_gdsl_bstree_write_func_t`  
Low-level binary search tree manipulation module, 33
  - `_gdsl_bstree_write_xml`  
Low-level binary search tree manipulation module, 46
- `_gdsl_list.h`, 252
- `_gdsl_list_alloc`  
Low-level doubly-linked list manipulation module, 49
- `_gdsl_list_dump`  
Low-level doubly-linked list manipulation module, 57
- `_gdsl_list_free`  
Low-level doubly-linked list manipulation module, 50
- `_gdsl_list_get_size`  
Low-level doubly-linked list manipulation module, 51
- `_gdsl_list_insert_after`  
Low-level doubly-linked list manipulation module, 52
- `_gdsl_list_insert_before`  
Low-level doubly-linked list manipulation module, 52
- `_gdsl_list_is_empty`  
Low-level doubly-linked list manipulation module, 50
- `_gdsl_list_link`  
Low-level doubly-linked list manipulation module, 51
- `_gdsl_list_map_backward`  
Low-level doubly-linked list manipulation module, 55
- `_gdsl_list_map_forward`  
Low-level doubly-linked list manipulation module, 54
- `_gdsl_list_remove`  
Low-level doubly-linked list manipulation module, 53
- `_gdsl_list_search`  
Low-level doubly-linked list manipulation module, 53
- `_gdsl_list_t`  
Low-level doubly-linked list manipulation module, 49
- `_gdsl_list_write`  
Low-level doubly-linked list manipulation module, 55
- `_gdsl_list_write_xml`  
Low-level doubly-linked list manipulation module, 56
- `_gdsl_node.h`, 253
- `_gdsl_node_alloc`  
Low-level doubly-linked node manipulation module, 61
- `_gdsl_node_dump`  
Low-level doubly-linked node manipulation module, 67
- `_gdsl_node_free`  
Low-level doubly-linked node manipulation module, 61
- `_gdsl_node_get_content`  
Low-level doubly-linked node manipulation module, 63
- `_gdsl_node_get_pred`  
Low-level doubly-linked node manipulation module, 62
- `_gdsl_node_get_succ`  
Low-level doubly-linked node manipulation module, 62
- `_gdsl_node_link`  
Low-level doubly-linked node manipulation module, 65
- `_gdsl_node_map_func_t`  
Low-level doubly-linked node manipulation module, 60
- `_gdsl_node_set_content`  
Low-level doubly-linked node manipulation module, 65
- `_gdsl_node_set_pred`  
Low-level doubly-linked node manipulation module, 64
- `_gdsl_node_set_succ`

- Low-level doubly-linked node manipulation module, 64
- `_gdsl_node_t`
  - Low-level doubly-linked node manipulation module, 60
- `_gdsl_node_unlink`
  - Low-level doubly-linked node manipulation module, 66
- `_gdsl_node_write`
  - Low-level doubly-linked node manipulation module, 66
- `_gdsl_node_write_func_t`
  - Low-level doubly-linked node manipulation module, 60
- `_gdsl_node_write_xml`
  - Low-level doubly-linked node manipulation module, 67
- 2D-Arrays manipulation module, 70
  - `gdsl_2darray_alloc`, 71
  - `gdsl_2darray_dump`, 78
  - `gdsl_2darray_free`, 72
  - `gdsl_2darray_get_columns_number`, 74
  - `gdsl_2darray_get_content`, 75
  - `gdsl_2darray_get_name`, 72
  - `gdsl_2darray_get_rows_number`, 73
  - `gdsl_2darray_get_size`, 74
  - `gdsl_2darray_set_content`, 76
  - `gdsl_2darray_set_name`, 75
  - `gdsl_2darray_t`, 71
  - `gdsl_2darray_write`, 77
  - `gdsl_2darray_write_xml`, 77
- `bool`
  - GDSL types, 246
- `gdsl.h`, 255
- `gdsl_2darray.h`, 255
- `gdsl_2darray_alloc`
  - 2D-Arrays manipulation module, 71
- `gdsl_2darray_dump`
  - 2D-Arrays manipulation module, 78
- `gdsl_2darray_free`
  - 2D-Arrays manipulation module, 72
- `gdsl_2darray_get_columns_number`
  - 2D-Arrays manipulation module, 74
- `gdsl_2darray_get_content`
  - 2D-Arrays manipulation module, 75
- `gdsl_2darray_get_name`
  - 2D-Arrays manipulation module, 72
- `gdsl_2darray_get_rows_number`
  - 2D-Arrays manipulation module, 73
- `gdsl_2darray_get_size`
  - 2D-Arrays manipulation module, 74
- `gdsl_2darray_set_content`
  - 2D-Arrays manipulation module, 76
- `gdsl_2darray_set_name`
  - 2D-Arrays manipulation module, 75
- `gdsl_2darray_t`
  - 2D-Arrays manipulation module, 71
- `gdsl_2darray_write`
  - 2D-Arrays manipulation module, 77
- `gdsl_2darray_write_xml`
  - 2D-Arrays manipulation module, 77
- `gdsl_alloc_func_t`
  - GDSL types, 243
- `gdsl_bstree.h`, 256
- `gdsl_bstree_alloc`
  - Binary search tree manipulation module, 81
- `gdsl_bstree_delete`
  - Binary search tree manipulation module, 88
- `gdsl_bstree_dump`
  - Binary search tree manipulation module, 94
- `gdsl_bstree_flush`
  - Binary search tree manipulation module, 83
- `gdsl_bstree_free`
  - Binary search tree manipulation module, 82
- `gdsl_bstree_get_height`
  - Binary search tree manipulation module, 86
- `gdsl_bstree_get_name`
  - Binary search tree manipulation module, 84
- `gdsl_bstree_get_root`
  - Binary search tree manipulation module, 85
- `gdsl_bstree_get_size`
  - Binary search tree manipulation module, 85
- `gdsl_bstree_insert`
  - Binary search tree manipulation module, 87
- `gdsl_bstree_is_empty`
  - Binary search tree manipulation module, 84

- gdsl\_bstree\_map\_infix
  - Binary search tree manipulation module, 91
- gdsl\_bstree\_map\_postfix
  - Binary search tree manipulation module, 91
- gdsl\_bstree\_map\_prefix
  - Binary search tree manipulation module, 90
- gdsl\_bstree\_remove
  - Binary search tree manipulation module, 88
- gdsl\_bstree\_search
  - Binary search tree manipulation module, 89
- gdsl\_bstree\_set\_name
  - Binary search tree manipulation module, 86
- gdsl\_bstree\_t
  - Binary search tree manipulation module, 81
- gdsl\_bstree\_write
  - Binary search tree manipulation module, 92
- gdsl\_bstree\_write\_xml
  - Binary search tree manipulation module, 93
- gdsl\_compare\_func\_t
  - GDSL types, 245
- gdsl\_constant\_t
  - GDSL types, 246
- gdsl\_copy\_func\_t
  - GDSL types, 244
- gdsl\_element\_t
  - GDSL types, 243
- gdsl\_free\_func\_t
  - GDSL types, 243
- gdsl\_get\_version
  - Main module, 69
- gdsl\_hash
  - Hashtable manipulation module, 97
- gdsl\_hash.h, 257
- gdsl\_hash\_alloc
  - Hashtable manipulation module, 98
- gdsl\_hash\_delete
  - Hashtable manipulation module, 106
- gdsl\_hash\_dump
  - Hashtable manipulation module, 110
- gdsl\_hash\_flush
  - Hashtable manipulation module, 99
- gdsl\_hash\_free
  - Hashtable manipulation module, 99
- gdsl\_hash\_func\_t
  - Hashtable manipulation module, 97
- gdsl\_hash\_get\_entries\_number
  - Hashtable manipulation module, 101
- gdsl\_hash\_get\_fill\_factor
  - Hashtable manipulation module, 103
- gdsl\_hash\_get\_lists\_max\_size
  - Hashtable manipulation module, 101
- gdsl\_hash\_get\_longest\_list\_size
  - Hashtable manipulation module, 102
- gdsl\_hash\_get\_name
  - Hashtable manipulation module, 100
- gdsl\_hash\_get\_size
  - Hashtable manipulation module, 102
- gdsl\_hash\_insert
  - Hashtable manipulation module, 104
- gdsl\_hash\_map
  - Hashtable manipulation module, 108
- gdsl\_hash\_modify
  - Hashtable manipulation module, 107
- gdsl\_hash\_remove
  - Hashtable manipulation module, 105
- gdsl\_hash\_search
  - Hashtable manipulation module, 107
- gdsl\_hash\_set\_name
  - Hashtable manipulation module, 104
- gdsl\_hash\_t
  - Hashtable manipulation module, 96
- gdsl\_hash\_write
  - Hashtable manipulation module, 109
- gdsl\_hash\_write\_xml
  - Hashtable manipulation module, 110
- gdsl\_heap.h, 259
- gdsl\_heap\_alloc
  - Heap manipulation module, 113
- gdsl\_heap\_delete\_top
  - Heap manipulation module, 120
- gdsl\_heap\_dump
  - Heap manipulation module, 123
- gdsl\_heap\_flush
  - Heap manipulation module, 115
- gdsl\_heap\_free
  - Heap manipulation module, 114
- gdsl\_heap\_get\_name
  - Heap manipulation module, 115
- gdsl\_heap\_get\_size
  - Heap manipulation module, 116
- gdsl\_heap\_get\_top

- Heap manipulation module, 116
- gdsl\_heap\_insert
  - Heap manipulation module, 119
- gdsl\_heap\_is\_empty
  - Heap manipulation module, 117
- gdsl\_heap\_map\_forward
  - Heap manipulation module, 121
- gdsl\_heap\_remove\_top
  - Heap manipulation module, 120
- gdsl\_heap\_set\_name
  - Heap manipulation module, 118
- gdsl\_heap\_set\_top
  - Heap manipulation module, 118
- gdsl\_heap\_t
  - Heap manipulation module, 113
- gdsl\_heap\_write
  - Heap manipulation module, 122
- gdsl\_heap\_write\_xml
  - Heap manipulation module, 122
- gdsl\_interval\_heap.h, 260
- gdsl\_interval\_heap\_alloc
  - Interval Heap manipulation module, 127
- gdsl\_interval\_heap\_delete\_max
  - Interval Heap manipulation module, 135
- gdsl\_interval\_heap\_delete\_min
  - Interval Heap manipulation module, 134
- gdsl\_interval\_heap\_dump
  - Interval Heap manipulation module, 138
- gdsl\_interval\_heap\_flush
  - Interval Heap manipulation module, 128
- gdsl\_interval\_heap\_free
  - Interval Heap manipulation module, 127
- gdsl\_interval\_heap\_get\_max
  - Interval Heap manipulation module, 134
- gdsl\_interval\_heap\_get\_min
  - Interval Heap manipulation module, 134
- gdsl\_interval\_heap\_get\_name
  - Interval Heap manipulation module, 129
- gdsl\_interval\_heap\_get\_size
  - Interval Heap manipulation module, 129
- gdsl\_interval\_heap\_insert
  - Interval Heap manipulation module, 131
- gdsl\_interval\_heap\_is\_empty
  - Interval Heap manipulation module, 130
- gdsl\_interval\_heap\_map\_forward
  - Interval Heap manipulation module, 136
- gdsl\_interval\_heap\_remove\_max
  - Interval Heap manipulation module, 132
- gdsl\_interval\_heap\_remove\_min
  - Interval Heap manipulation module, 133
- gdsl\_interval\_heap\_set\_max\_size
  - Interval Heap manipulation module, 130
- gdsl\_interval\_heap\_set\_name
  - Interval Heap manipulation module, 131
- gdsl\_interval\_heap\_t
  - Interval Heap manipulation module, 126
- gdsl\_interval\_heap\_write
  - Interval Heap manipulation module, 136
- gdsl\_interval\_heap\_write\_xml
  - Interval Heap manipulation module, 137
- gdsl\_key\_func\_t
  - Hashtable manipulation module, 96
- gdsl\_list.h, 261
- gdsl\_list\_alloc
  - Doubly-linked list manipulation module, 142
- gdsl\_list\_cursor\_alloc
  - Doubly-linked list manipulation module, 161
- gdsl\_list\_cursor\_delete
  - Doubly-linked list manipulation module, 172
- gdsl\_list\_cursor\_delete\_after
  - Doubly-linked list manipulation module, 173
- gdsl\_list\_cursor\_delete\_before
  - Doubly-linked list manipulation module, 173
- gdsl\_list\_cursor\_free

- Doubly-linked list manipulation module, 161
- gdsl\_list\_cursor\_get\_content
  - Doubly-linked list manipulation module, 168
- gdsl\_list\_cursor\_has\_pred
  - Doubly-linked list manipulation module, 167
- gdsl\_list\_cursor\_has\_succ
  - Doubly-linked list manipulation module, 166
- gdsl\_list\_cursor\_insert\_after
  - Doubly-linked list manipulation module, 169
- gdsl\_list\_cursor\_insert\_before
  - Doubly-linked list manipulation module, 169
- gdsl\_list\_cursor\_is\_on\_head
  - Doubly-linked list manipulation module, 165
- gdsl\_list\_cursor\_is\_on\_tail
  - Doubly-linked list manipulation module, 166
- gdsl\_list\_cursor\_move\_to\_head
  - Doubly-linked list manipulation module, 162
- gdsl\_list\_cursor\_move\_to\_position
  - Doubly-linked list manipulation module, 164
- gdsl\_list\_cursor\_move\_to\_tail
  - Doubly-linked list manipulation module, 162
- gdsl\_list\_cursor\_move\_to\_value
  - Doubly-linked list manipulation module, 163
- gdsl\_list\_cursor\_remove
  - Doubly-linked list manipulation module, 170
- gdsl\_list\_cursor\_remove\_after
  - Doubly-linked list manipulation module, 171
- gdsl\_list\_cursor\_remove\_before
  - Doubly-linked list manipulation module, 172
- gdsl\_list\_cursor\_set\_content
  - Doubly-linked list manipulation module, 168
- gdsl\_list\_cursor\_step\_backward
  - Doubly-linked list manipulation module, 165
- gdsl\_list\_cursor\_step\_forward
  - Doubly-linked list manipulation module, 164
- gdsl\_list\_cursor\_t
  - Doubly-linked list manipulation module, 142
- gdsl\_list\_delete
  - Doubly-linked list manipulation module, 153
- gdsl\_list\_delete\_head
  - Doubly-linked list manipulation module, 151
- gdsl\_list\_delete\_tail
  - Doubly-linked list manipulation module, 152
- gdsl\_list\_dump
  - Doubly-linked list manipulation module, 160
- gdsl\_list\_flush
  - Doubly-linked list manipulation module, 144
- gdsl\_list\_free
  - Doubly-linked list manipulation module, 143
- gdsl\_list\_get\_head
  - Doubly-linked list manipulation module, 146
- gdsl\_list\_get\_name
  - Doubly-linked list manipulation module, 144
- gdsl\_list\_get\_size
  - Doubly-linked list manipulation module, 145
- gdsl\_list\_get\_tail
  - Doubly-linked list manipulation module, 146
- gdsl\_list\_insert\_head
  - Doubly-linked list manipulation module, 148
- gdsl\_list\_insert\_tail
  - Doubly-linked list manipulation module, 148
- gdsl\_list\_is\_empty
  - Doubly-linked list manipulation module, 145
- gdsl\_list\_map\_backward
  - Doubly-linked list manipulation module, 158
- gdsl\_list\_map\_forward

- Doubly-linked list manipulation module, 157
- gdsl\_list\_remove
  - Doubly-linked list manipulation module, 151
- gdsl\_list\_remove\_head
  - Doubly-linked list manipulation module, 149
- gdsl\_list\_remove\_tail
  - Doubly-linked list manipulation module, 150
- gdsl\_list\_search
  - Doubly-linked list manipulation module, 153
- gdsl\_list\_search\_by\_position
  - Doubly-linked list manipulation module, 154
- gdsl\_list\_search\_max
  - Doubly-linked list manipulation module, 155
- gdsl\_list\_search\_min
  - Doubly-linked list manipulation module, 156
- gdsl\_list\_set\_name
  - Doubly-linked list manipulation module, 147
- gdsl\_list\_sort
  - Doubly-linked list manipulation module, 156
- gdsl\_list\_t
  - Doubly-linked list manipulation module, 142
- gdsl\_list\_write
  - Doubly-linked list manipulation module, 158
- gdsl\_list\_write\_xml
  - Doubly-linked list manipulation module, 159
- gdsl\_location\_t
  - GDSL types, 246
- gdsl\_macros.h, 264
- gdsl\_map\_func\_t
  - GDSL types, 244
- gdsl\_perm.h, 264
- gdsl\_perm\_alloc
  - Permutation manipulation module, 180
- gdsl\_perm\_apply\_on\_array
  - Permutation manipulation module, 192
- gdsl\_perm\_canonical\_cycles\_count
  - Permutation manipulation module, 185
- gdsl\_perm\_canonical\_to\_linear
  - Permutation manipulation module, 189
- gdsl\_perm\_copy
  - Permutation manipulation module, 181
- gdsl\_perm\_data\_t
  - Permutation manipulation module, 179
- gdsl\_perm\_dump
  - Permutation manipulation module, 194
- gdsl\_perm\_free
  - Permutation manipulation module, 180
- gdsl\_perm\_get\_element
  - Permutation manipulation module, 183
- gdsl\_perm\_get\_elements\_array
  - Permutation manipulation module, 183
- gdsl\_perm\_get\_name
  - Permutation manipulation module, 181
- gdsl\_perm\_get\_size
  - Permutation manipulation module, 182
- gdsl\_perm\_inverse
  - Permutation manipulation module, 190
- gdsl\_perm\_linear\_cycles\_count
  - Permutation manipulation module, 184
- gdsl\_perm\_linear\_inversions\_count
  - Permutation manipulation module, 184
- gdsl\_perm\_linear\_next
  - Permutation manipulation module, 186
- gdsl\_perm\_linear\_prev
  - Permutation manipulation module, 187
- gdsl\_perm\_linear\_to\_canonical
  - Permutation manipulation module, 189
- gdsl\_perm\_multiply



- Permutation manipulation module, 188
- gdsl\_perm\_position\_t
  - Permutation manipulation module, 179
- gdsl\_perm\_randomize
  - Permutation manipulation module, 191
- gdsl\_perm\_reverse
  - Permutation manipulation module, 191
- gdsl\_perm\_set\_elements\_array
  - Permutation manipulation module, 187
- gdsl\_perm\_set\_name
  - Permutation manipulation module, 186
- gdsl\_perm\_t
  - Permutation manipulation module, 179
- gdsl\_perm\_write
  - Permutation manipulation module, 192
- gdsl\_perm\_write\_func\_t
  - Permutation manipulation module, 179
- gdsl\_perm\_write\_xml
  - Permutation manipulation module, 193
- gdsl\_queue.h, 266
- gdsl\_queue\_alloc
  - Queue manipulation module, 196
- gdsl\_queue\_dump
  - Queue manipulation module, 207
- gdsl\_queue\_flush
  - Queue manipulation module, 198
- gdsl\_queue\_free
  - Queue manipulation module, 197
- gdsl\_queue\_get\_head
  - Queue manipulation module, 200
- gdsl\_queue\_get\_name
  - Queue manipulation module, 198
- gdsl\_queue\_get\_size
  - Queue manipulation module, 199
- gdsl\_queue\_get\_tail
  - Queue manipulation module, 201
- gdsl\_queue\_insert
  - Queue manipulation module, 202
- gdsl\_queue\_is\_empty
  - Queue manipulation module, 199
- gdsl\_queue\_map\_backward
  - Queue manipulation module, 205
- gdsl\_queue\_map\_forward
  - Queue manipulation module, 205
- gdsl\_queue\_remove
  - Queue manipulation module, 203
- gdsl\_queue\_search
  - Queue manipulation module, 203
- gdsl\_queue\_search\_by\_position
  - Queue manipulation module, 204
- gdsl\_queue\_set\_name
  - Queue manipulation module, 201
- gdsl\_queue\_t
  - Queue manipulation module, 196
- gdsl\_queue\_write
  - Queue manipulation module, 206
- gdsl\_queue\_write\_xml
  - Queue manipulation module, 207
- gdsl\_rbtrees.h, 267
- gdsl\_rbtrees\_alloc
  - Red-black tree manipulation module, 210
- gdsl\_rbtrees\_delete
  - Red-black tree manipulation module, 217
- gdsl\_rbtrees\_dump
  - Red-black tree manipulation module, 223
- gdsl\_rbtrees\_flush
  - Red-black tree manipulation module, 212
- gdsl\_rbtrees\_free
  - Red-black tree manipulation module, 211
- gdsl\_rbtrees\_get\_name
  - Red-black tree manipulation module, 212
- gdsl\_rbtrees\_get\_root
  - Red-black tree manipulation module, 214
- gdsl\_rbtrees\_get\_size
  - Red-black tree manipulation module, 214
- gdsl\_rbtrees\_height
  - Red-black tree manipulation module, 215
- gdsl\_rbtrees\_insert
  - Red-black tree manipulation module, 216
- gdsl\_rbtrees\_is\_empty

- Red-black tree manipulation module, 213
- gdsl\_rbtrees\_map\_infix
  - Red-black tree manipulation module, 220
- gdsl\_rbtrees\_map\_postfix
  - Red-black tree manipulation module, 221
- gdsl\_rbtrees\_map\_prefix
  - Red-black tree manipulation module, 219
- gdsl\_rbtrees\_remove
  - Red-black tree manipulation module, 217
- gdsl\_rbtrees\_search
  - Red-black tree manipulation module, 218
- gdsl\_rbtrees\_set\_name
  - Red-black tree manipulation module, 215
- gdsl\_rbtrees\_t
  - Red-black tree manipulation module, 210
- gdsl\_rbtrees\_write
  - Red-black tree manipulation module, 221
- gdsl\_rbtrees\_write\_xml
  - Red-black tree manipulation module, 222
- gdsl\_sort
  - Sort module, 225
- gdsl\_sort.h, 269
- gdsl\_stack.h, 269
- gdsl\_stack\_alloc
  - Stack manipulation module, 229
- gdsl\_stack\_dump
  - Stack manipulation module, 241
- gdsl\_stack\_flush
  - Stack manipulation module, 230
- gdsl\_stack\_free
  - Stack manipulation module, 229
- gdsl\_stack\_get\_bottom
  - Stack manipulation module, 233
- gdsl\_stack\_get\_growing\_factor
  - Stack manipulation module, 232
- gdsl\_stack\_get\_name
  - Stack manipulation module, 230
- gdsl\_stack\_get\_size
  - Stack manipulation module, 231
- gdsl\_stack\_get\_top
  - Stack manipulation module, 233
- gdsl\_stack\_insert
  - Stack manipulation module, 235
- gdsl\_stack\_is\_empty
  - Stack manipulation module, 232
- gdsl\_stack\_map\_backward
  - Stack manipulation module, 239
- gdsl\_stack\_map\_forward
  - Stack manipulation module, 238
- gdsl\_stack\_remove
  - Stack manipulation module, 236
- gdsl\_stack\_search
  - Stack manipulation module, 237
- gdsl\_stack\_search\_by\_position
  - Stack manipulation module, 237
- gdsl\_stack\_set\_growing\_factor
  - Stack manipulation module, 235
- gdsl\_stack\_set\_name
  - Stack manipulation module, 234
- gdsl\_stack\_t
  - Stack manipulation module, 228
- gdsl\_stack\_write
  - Stack manipulation module, 239
- gdsl\_stack\_write\_xml
  - Stack manipulation module, 240
- gdsl\_types.h, 270
- gdsl\_write\_func\_t
  - GDSL types, 245
- mainpage.h, 271
- ulong
  - GDSL types, 245
- ushort
  - GDSL types, 245