

gdsI

1.8

Generated by Doxygen 1.7.6.1

Fri Jun 15 2018 12:17:12

Contents

1	gdsl	1
1.1	Introduction	1
1.2	About	1
1.3	Copyright	1
1.4	Project Manager	1
1.5	Authors	2
1.6	Thanks	2
1.7	GNU Free Documentation License	3
2	Module Index	11
2.1	Modules	11
3	File Index	13
3.1	File List	13
4	Module Documentation	15
4.1	Low level binary tree manipulation module.	15
4.1.1	Detailed Description	17
4.1.2	Typedef Documentation	17
4.1.2.1	_gdsl_bintree_t	17
4.1.2.2	_gdsl_bintree_map_func_t	17
4.1.2.3	_gdsl_bintree_write_func_t	18
4.1.3	Function Documentation	18
4.1.3.1	_gdsl_bintree_alloc	18
4.1.3.2	_gdsl_bintree_free	19
4.1.3.3	_gdsl_bintree_copy	19

4.1.3.4	<code>_gdsl_bintree_is_empty</code>	20
4.1.3.5	<code>_gdsl_bintree_is_leaf</code>	21
4.1.3.6	<code>_gdsl_bintree_is_root</code>	21
4.1.3.7	<code>_gdsl_bintree_get_content</code>	22
4.1.3.8	<code>_gdsl_bintree_get_parent</code>	22
4.1.3.9	<code>_gdsl_bintree_get_left</code>	23
4.1.3.10	<code>_gdsl_bintree_get_right</code>	24
4.1.3.11	<code>_gdsl_bintree_get_left_ref</code>	24
4.1.3.12	<code>_gdsl_bintree_get_right_ref</code>	25
4.1.3.13	<code>_gdsl_bintree_get_height</code>	25
4.1.3.14	<code>_gdsl_bintree_get_size</code>	26
4.1.3.15	<code>_gdsl_bintree_set_content</code>	26
4.1.3.16	<code>_gdsl_bintree_set_parent</code>	27
4.1.3.17	<code>_gdsl_bintree_set_left</code>	27
4.1.3.18	<code>_gdsl_bintree_set_right</code>	28
4.1.3.19	<code>_gdsl_bintree_rotate_left</code>	28
4.1.3.20	<code>_gdsl_bintree_rotate_right</code>	29
4.1.3.21	<code>_gdsl_bintree_rotate_left_right</code>	30
4.1.3.22	<code>_gdsl_bintree_rotate_right_left</code>	30
4.1.3.23	<code>_gdsl_bintree_map_prefix</code>	31
4.1.3.24	<code>_gdsl_bintree_map_infix</code>	32
4.1.3.25	<code>_gdsl_bintree_map_postfix</code>	33
4.1.3.26	<code>_gdsl_bintree_write</code>	33
4.1.3.27	<code>_gdsl_bintree_write_xml</code>	34
4.1.3.28	<code>_gdsl_bintree_dump</code>	35
4.2	Low-level binary search tree manipulation module.	36
4.2.1	Detailed Description	37
4.2.2	Typedef Documentation	37
4.2.2.1	<code>_gdsl_bstree_t</code>	37
4.2.2.2	<code>_gdsl_bstree_map_func_t</code>	38
4.2.2.3	<code>_gdsl_bstree_write_func_t</code>	38
4.2.3	Function Documentation	38
4.2.3.1	<code>_gdsl_bstree_alloc</code>	38
4.2.3.2	<code>_gdsl_bstree_free</code>	39

4.2.3.3	<code>_gdsl_bstree_copy</code>	40
4.2.3.4	<code>_gdsl_bstree_is_empty</code>	40
4.2.3.5	<code>_gdsl_bstree_is_leaf</code>	41
4.2.3.6	<code>_gdsl_bstree_get_content</code>	41
4.2.3.7	<code>_gdsl_bstree_is_root</code>	42
4.2.3.8	<code>_gdsl_bstree_get_parent</code>	42
4.2.3.9	<code>_gdsl_bstree_get_left</code>	43
4.2.3.10	<code>_gdsl_bstree_get_right</code>	44
4.2.3.11	<code>_gdsl_bstree_get_size</code>	44
4.2.3.12	<code>_gdsl_bstree_get_height</code>	45
4.2.3.13	<code>_gdsl_bstree_insert</code>	45
4.2.3.14	<code>_gdsl_bstree_remove</code>	46
4.2.3.15	<code>_gdsl_bstree_search</code>	47
4.2.3.16	<code>_gdsl_bstree_search_next</code>	47
4.2.3.17	<code>_gdsl_bstree_map_prefix</code>	48
4.2.3.18	<code>_gdsl_bstree_map_infix</code>	49
4.2.3.19	<code>_gdsl_bstree_map_postfix</code>	50
4.2.3.20	<code>_gdsl_bstree_write</code>	50
4.2.3.21	<code>_gdsl_bstree_write_xml</code>	51
4.2.3.22	<code>_gdsl_bstree_dump</code>	52
4.3	Low-level doubly-linked list manipulation module.	53
4.3.1	Detailed Description	54
4.3.2	Typedef Documentation	54
4.3.2.1	<code>_gdsl_list_t</code>	54
4.3.3	Function Documentation	54
4.3.3.1	<code>_gdsl_list_alloc</code>	54
4.3.3.2	<code>_gdsl_list_free</code>	55
4.3.3.3	<code>_gdsl_list_is_empty</code>	55
4.3.3.4	<code>_gdsl_list_get_size</code>	56
4.3.3.5	<code>_gdsl_list_link</code>	56
4.3.3.6	<code>_gdsl_list_insert_after</code>	57
4.3.3.7	<code>_gdsl_list_insert_before</code>	57
4.3.3.8	<code>_gdsl_list_remove</code>	58
4.3.3.9	<code>_gdsl_list_search</code>	58

4.3.3.10	<code>_gdsl_list_map_forward</code>	59
4.3.3.11	<code>_gdsl_list_map_backward</code>	59
4.3.3.12	<code>_gdsl_list_write</code>	60
4.3.3.13	<code>_gdsl_list_write_xml</code>	61
4.3.3.14	<code>_gdsl_list_dump</code>	62
4.4	Low-level doubly-linked node manipulation module.	63
4.4.1	Detailed Description	64
4.4.2	Typedef Documentation	64
4.4.2.1	<code>_gdsl_node_t</code>	64
4.4.2.2	<code>_gdsl_node_map_func_t</code>	64
4.4.2.3	<code>_gdsl_node_write_func_t</code>	64
4.4.3	Function Documentation	65
4.4.3.1	<code>_gdsl_node_alloc</code>	65
4.4.3.2	<code>_gdsl_node_free</code>	65
4.4.3.3	<code>_gdsl_node_get_succ</code>	66
4.4.3.4	<code>_gdsl_node_get_pred</code>	66
4.4.3.5	<code>_gdsl_node_get_content</code>	67
4.4.3.6	<code>_gdsl_node_set_succ</code>	67
4.4.3.7	<code>_gdsl_node_set_pred</code>	68
4.4.3.8	<code>_gdsl_node_set_content</code>	68
4.4.3.9	<code>_gdsl_node_link</code>	69
4.4.3.10	<code>_gdsl_node_unlink</code>	69
4.4.3.11	<code>_gdsl_node_write</code>	70
4.4.3.12	<code>_gdsl_node_write_xml</code>	71
4.4.3.13	<code>_gdsl_node_dump</code>	71
4.5	Main module	73
4.5.1	Detailed Description	73
4.5.2	Function Documentation	73
4.5.2.1	<code>gdsl_get_version</code>	73
4.6	2D-Arrays manipulation module.	74
4.6.1	Detailed Description	75
4.6.2	Typedef Documentation	75
4.6.2.1	<code>gdsl_2darray_t</code>	75
4.6.3	Function Documentation	75

4.6.3.1	gdsl_2darray_alloc	75
4.6.3.2	gdsl_2darray_free	76
4.6.3.3	gdsl_2darray_get_name	76
4.6.3.4	gdsl_2darray_get_rows_number	77
4.6.3.5	gdsl_2darray_get_columns_number	78
4.6.3.6	gdsl_2darray_get_size	78
4.6.3.7	gdsl_2darray_get_content	79
4.6.3.8	gdsl_2darray_set_name	80
4.6.3.9	gdsl_2darray_set_content	80
4.6.3.10	gdsl_2darray_write	81
4.6.3.11	gdsl_2darray_write_xml	82
4.6.3.12	gdsl_2darray_dump	82
4.7	Binary search tree manipulation module.	84
4.7.1	Detailed Description	85
4.7.2	Typedef Documentation	85
4.7.2.1	gdsl_bstree_t	85
4.7.3	Function Documentation	85
4.7.3.1	gdsl_bstree_alloc	85
4.7.3.2	gdsl_bstree_free	86
4.7.3.3	gdsl_bstree_flush	87
4.7.3.4	gdsl_bstree_get_name	88
4.7.3.5	gdsl_bstree_is_empty	88
4.7.3.6	gdsl_bstree_get_root	89
4.7.3.7	gdsl_bstree_get_size	89
4.7.3.8	gdsl_bstree_get_height	90
4.7.3.9	gdsl_bstree_set_name	90
4.7.3.10	gdsl_bstree_insert	91
4.7.3.11	gdsl_bstree_remove	92
4.7.3.12	gdsl_bstree_delete	93
4.7.3.13	gdsl_bstree_search	93
4.7.3.14	gdsl_bstree_map_prefix	94
4.7.3.15	gdsl_bstree_map_infix	95
4.7.3.16	gdsl_bstree_map_postfix	96
4.7.3.17	gdsl_bstree_write	96

4.7.3.18	gdsl_bstree_write_xml	97
4.7.3.19	gdsl_bstree_dump	98
4.8	Hashtable manipulation module.	99
4.8.1	Detailed Description	100
4.8.2	Typedef Documentation	100
4.8.2.1	gdsl_hash_t	100
4.8.2.2	gdsl_key_func_t	100
4.8.2.3	gdsl_hash_func_t	101
4.8.3	Function Documentation	101
4.8.3.1	gdsl_hash	101
4.8.3.2	gdsl_hash_alloc	102
4.8.3.3	gdsl_hash_free	103
4.8.3.4	gdsl_hash_flush	103
4.8.3.5	gdsl_hash_get_name	104
4.8.3.6	gdsl_hash_get_entries_number	105
4.8.3.7	gdsl_hash_get_lists_max_size	105
4.8.3.8	gdsl_hash_get_longest_list_size	106
4.8.3.9	gdsl_hash_get_size	106
4.8.3.10	gdsl_hash_get_fill_factor	107
4.8.3.11	gdsl_hash_set_name	108
4.8.3.12	gdsl_hash_insert	108
4.8.3.13	gdsl_hash_remove	109
4.8.3.14	gdsl_hash_delete	110
4.8.3.15	gdsl_hash_modify	111
4.8.3.16	gdsl_hash_search	112
4.8.3.17	gdsl_hash_map	112
4.8.3.18	gdsl_hash_write	113
4.8.3.19	gdsl_hash_write_xml	114
4.8.3.20	gdsl_hash_dump	114
4.9	Heap manipulation module.	116
4.9.1	Detailed Description	117
4.9.2	Typedef Documentation	117
4.9.2.1	gdsl_heap_t	117
4.9.3	Function Documentation	117

4.9.3.1	gdsl_heap_alloc	117
4.9.3.2	gdsl_heap_free	118
4.9.3.3	gdsl_heap_flush	119
4.9.3.4	gdsl_heap_get_name	119
4.9.3.5	gdsl_heap_get_size	120
4.9.3.6	gdsl_heap_get_top	120
4.9.3.7	gdsl_heap_is_empty	121
4.9.3.8	gdsl_heap_set_name	121
4.9.3.9	gdsl_heap_set_top	122
4.9.3.10	gdsl_heap_insert	123
4.9.3.11	gdsl_heap_remove_top	123
4.9.3.12	gdsl_heap_delete_top	124
4.9.3.13	gdsl_heap_map_forward	125
4.9.3.14	gdsl_heap_write	125
4.9.3.15	gdsl_heap_write_xml	126
4.9.3.16	gdsl_heap_dump	127
4.10	Interval Heap manipulation module.	128
4.10.1	Detailed Description	129
4.10.2	Typedef Documentation	129
4.10.2.1	gdsl_interval_heap_t	129
4.10.3	Function Documentation	129
4.10.3.1	gdsl_interval_heap_alloc	129
4.10.3.2	gdsl_interval_heap_free	130
4.10.3.3	gdsl_interval_heap_flush	131
4.10.3.4	gdsl_interval_heap_get_name	132
4.10.3.5	gdsl_interval_heap_get_size	132
4.10.3.6	gdsl_interval_heap_set_max_size	133
4.10.3.7	gdsl_interval_heap_is_empty	133
4.10.3.8	gdsl_interval_heap_set_name	134
4.10.3.9	gdsl_interval_heap_insert	134
4.10.3.10	gdsl_interval_heap_remove_max	135
4.10.3.11	gdsl_interval_heap_remove_min	136
4.10.3.12	gdsl_interval_heap_get_min	137
4.10.3.13	gdsl_interval_heap_get_max	137

4.10.3.14	gdsl_interval_heap_delete_min	137
4.10.3.15	gdsl_interval_heap_delete_max	138
4.10.3.16	gdsl_interval_heap_map_forward	139
4.10.3.17	gdsl_interval_heap_write	139
4.10.3.18	gdsl_interval_heap_write_xml	140
4.10.3.19	gdsl_interval_heap_dump	141
4.11	Doubly-linked list manipulation module.	142
4.11.1	Detailed Description	145
4.11.2	Typedef Documentation	145
4.11.2.1	gdsl_list_t	145
4.11.2.2	gdsl_list_cursor_t	145
4.11.3	Function Documentation	145
4.11.3.1	gdsl_list_alloc	145
4.11.3.2	gdsl_list_free	146
4.11.3.3	gdsl_list_flush	147
4.11.3.4	gdsl_list_get_name	147
4.11.3.5	gdsl_list_get_size	148
4.11.3.6	gdsl_list_is_empty	148
4.11.3.7	gdsl_list_get_head	149
4.11.3.8	gdsl_list_get_tail	149
4.11.3.9	gdsl_list_set_name	150
4.11.3.10	gdsl_list_insert_head	151
4.11.3.11	gdsl_list_insert_tail	151
4.11.3.12	gdsl_list_remove_head	152
4.11.3.13	gdsl_list_remove_tail	153
4.11.3.14	gdsl_list_remove	153
4.11.3.15	gdsl_list_delete_head	154
4.11.3.16	gdsl_list_delete_tail	155
4.11.3.17	gdsl_list_delete	155
4.11.3.18	gdsl_list_search	156
4.11.3.19	gdsl_list_search_by_position	157
4.11.3.20	gdsl_list_search_max	158
4.11.3.21	gdsl_list_search_min	159
4.11.3.22	gdsl_list_sort	159

4.11.3.23	gdsl_list_map_forward	160
4.11.3.24	gdsl_list_map_backward	161
4.11.3.25	gdsl_list_write	161
4.11.3.26	gdsl_list_write_xml	162
4.11.3.27	gdsl_list_dump	163
4.11.3.28	gdsl_list_cursor_alloc	163
4.11.3.29	gdsl_list_cursor_free	164
4.11.3.30	gdsl_list_cursor_move_to_head	165
4.11.3.31	gdsl_list_cursor_move_to_tail	165
4.11.3.32	gdsl_list_cursor_move_to_value	166
4.11.3.33	gdsl_list_cursor_move_to_position	166
4.11.3.34	gdsl_list_cursor_step_forward	167
4.11.3.35	gdsl_list_cursor_step_backward	167
4.11.3.36	gdsl_list_cursor_is_on_head	168
4.11.3.37	gdsl_list_cursor_is_on_tail	169
4.11.3.38	gdsl_list_cursor_has_succ	169
4.11.3.39	gdsl_list_cursor_has_pred	170
4.11.3.40	gdsl_list_cursor_set_content	170
4.11.3.41	gdsl_list_cursor_get_content	171
4.11.3.42	gdsl_list_cursor_insert_after	171
4.11.3.43	gdsl_list_cursor_insert_before	172
4.11.3.44	gdsl_list_cursor_remove	173
4.11.3.45	gdsl_list_cursor_remove_after	174
4.11.3.46	gdsl_list_cursor_remove_before	174
4.11.3.47	gdsl_list_cursor_delete	175
4.11.3.48	gdsl_list_cursor_delete_after	175
4.11.3.49	gdsl_list_cursor_delete_before	176
4.12	Various macros module.	178
4.12.1	Detailed Description	178
4.12.2	Define Documentation	178
4.12.2.1	GDSL_MAX	178
4.12.2.2	GDSL_MIN	179
4.13	Permutation manipulation module.	180
4.13.1	Detailed Description	181

4.13.2	Typedef Documentation	182
4.13.2.1	gdsI_perm_t	182
4.13.2.2	gdsI_perm_write_func_t	182
4.13.2.3	gdsI_perm_data_t	182
4.13.3	Enumeration Type Documentation	182
4.13.3.1	gdsI_perm_position_t	182
4.13.4	Function Documentation	182
4.13.4.1	gdsI_perm_alloc	183
4.13.4.2	gdsI_perm_free	183
4.13.4.3	gdsI_perm_copy	184
4.13.4.4	gdsI_perm_get_name	185
4.13.4.5	gdsI_perm_get_size	185
4.13.4.6	gdsI_perm_get_element	186
4.13.4.7	gdsI_perm_get_elements_array	186
4.13.4.8	gdsI_perm_linear_inversions_count	187
4.13.4.9	gdsI_perm_linear_cycles_count	187
4.13.4.10	gdsI_perm_canonical_cycles_count	188
4.13.4.11	gdsI_perm_set_name	189
4.13.4.12	gdsI_perm_linear_next	189
4.13.4.13	gdsI_perm_linear_prev	190
4.13.4.14	gdsI_perm_set_elements_array	191
4.13.4.15	gdsI_perm_multiply	191
4.13.4.16	gdsI_perm_linear_to_canonical	192
4.13.4.17	gdsI_perm_canonical_to_linear	192
4.13.4.18	gdsI_perm_inverse	193
4.13.4.19	gdsI_perm_reverse	194
4.13.4.20	gdsI_perm_randomize	194
4.13.4.21	gdsI_perm_apply_on_array	195
4.13.4.22	gdsI_perm_write	196
4.13.4.23	gdsI_perm_write_xml	196
4.13.4.24	gdsI_perm_dump	197
4.14	Queue manipulation module.	198
4.14.1	Detailed Description	199
4.14.2	Typedef Documentation	199

4.14.2.1	gdsl_queue_t	199
4.14.3	Function Documentation	199
4.14.3.1	gdsl_queue_alloc	199
4.14.3.2	gdsl_queue_free	200
4.14.3.3	gdsl_queue_flush	201
4.14.3.4	gdsl_queue_get_name	201
4.14.3.5	gdsl_queue_get_size	202
4.14.3.6	gdsl_queue_is_empty	202
4.14.3.7	gdsl_queue_get_head	203
4.14.3.8	gdsl_queue_get_tail	204
4.14.3.9	gdsl_queue_set_name	204
4.14.3.10	gdsl_queue_insert	205
4.14.3.11	gdsl_queue_remove	205
4.14.3.12	gdsl_queue_search	206
4.14.3.13	gdsl_queue_search_by_position	207
4.14.3.14	gdsl_queue_map_forward	207
4.14.3.15	gdsl_queue_map_backward	208
4.14.3.16	gdsl_queue_write	209
4.14.3.17	gdsl_queue_write_xml	209
4.14.3.18	gdsl_queue_dump	210
4.15	Red-black tree manipulation module.	212
4.15.1	Detailed Description	213
4.15.2	Typedef Documentation	213
4.15.2.1	gdsl_rbtrees_t	213
4.15.3	Function Documentation	213
4.15.3.1	gdsl_rbtrees_alloc	213
4.15.3.2	gdsl_rbtrees_free	214
4.15.3.3	gdsl_rbtrees_flush	215
4.15.3.4	gdsl_rbtrees_get_name	215
4.15.3.5	gdsl_rbtrees_is_empty	216
4.15.3.6	gdsl_rbtrees_get_root	216
4.15.3.7	gdsl_rbtrees_get_size	217
4.15.3.8	gdsl_rbtrees_height	218
4.15.3.9	gdsl_rbtrees_set_name	218

4.15.3.10	gdsI_rbtree_insert	219
4.15.3.11	gdsI_rbtree_remove	220
4.15.3.12	gdsI_rbtree_delete	220
4.15.3.13	gdsI_rbtree_search	221
4.15.3.14	gdsI_rbtree_map_prefix	222
4.15.3.15	gdsI_rbtree_map_infix	223
4.15.3.16	gdsI_rbtree_map_postfix	223
4.15.3.17	gdsI_rbtree_write	224
4.15.3.18	gdsI_rbtree_write_xml	225
4.15.3.19	gdsI_rbtree_dump	226
4.16	Sort module.	227
4.16.1	Detailed Description	227
4.16.2	Function Documentation	227
4.16.2.1	gdsI_sort	227
4.17	Stack manipulation module.	228
4.17.1	Detailed Description	229
4.17.2	Typedef Documentation	229
4.17.2.1	gdsI_stack_t	229
4.17.3	Function Documentation	229
4.17.3.1	gdsI_stack_alloc	229
4.17.3.2	gdsI_stack_free	230
4.17.3.3	gdsI_stack_flush	231
4.17.3.4	gdsI_stack_get_name	231
4.17.3.5	gdsI_stack_get_size	232
4.17.3.6	gdsI_stack_get_growing_factor	232
4.17.3.7	gdsI_stack_is_empty	233
4.17.3.8	gdsI_stack_get_top	234
4.17.3.9	gdsI_stack_get_bottom	234
4.17.3.10	gdsI_stack_set_name	235
4.17.3.11	gdsI_stack_set_growing_factor	236
4.17.3.12	gdsI_stack_insert	236
4.17.3.13	gdsI_stack_remove	237
4.17.3.14	gdsI_stack_search	238
4.17.3.15	gdsI_stack_search_by_position	238

4.17.3.16	gdsl_stack_map_forward	239
4.17.3.17	gdsl_stack_map_backward	240
4.17.3.18	gdsl_stack_write	240
4.17.3.19	gdsl_stack_write_xml	241
4.17.3.20	gdsl_stack_dump	242
4.18	GDSL types.	243
4.18.1	Detailed Description	243
4.18.2	Typedef Documentation	244
4.18.2.1	gdsl_element_t	244
4.18.2.2	gdsl_alloc_func_t	244
4.18.2.3	gdsl_free_func_t	244
4.18.2.4	gdsl_copy_func_t	245
4.18.2.5	gdsl_map_func_t	245
4.18.2.6	gdsl_compare_func_t	245
4.18.2.7	gdsl_write_func_t	246
4.18.2.8	ulong	246
4.18.2.9	ushort	246
4.18.3	Enumeration Type Documentation	246
4.18.3.1	gdsl_constant_t	247
4.18.3.2	gdsl_location_t	247
4.18.3.3	bool	247
5	File Documentation	249
5.1	_gdsl_bintree.h File Reference	249
5.1.1	Detailed Description	251
5.2	_gdsl_bstree.h File Reference	251
5.2.1	Detailed Description	253
5.3	_gdsl_list.h File Reference	253
5.3.1	Detailed Description	254
5.4	_gdsl_node.h File Reference	254
5.4.1	Detailed Description	255
5.5	gdsl.h File Reference	255
5.6	gdsl_2darray.h File Reference	255
5.6.1	Detailed Description	256

5.7	gdsI_bstree.h File Reference	257
5.7.1	Detailed Description	258
5.8	gdsI_hash.h File Reference	258
5.8.1	Detailed Description	260
5.9	gdsI_heap.h File Reference	260
5.9.1	Detailed Description	261
5.10	gdsI_interval_heap.h File Reference	261
5.10.1	Detailed Description	262
5.11	gdsI_list.h File Reference	263
5.11.1	Detailed Description	266
5.12	gdsI_macros.h File Reference	266
5.12.1	Detailed Description	266
5.13	gdsI_perm.h File Reference	266
5.13.1	Detailed Description	268
5.14	gdsI_queue.h File Reference	268
5.14.1	Detailed Description	269
5.15	gdsI_rbtrees.h File Reference	269
5.15.1	Detailed Description	271
5.16	gdsI_sort.h File Reference	271
5.16.1	Detailed Description	271
5.17	gdsI_stack.h File Reference	271
5.17.1	Detailed Description	273
5.18	gdsI_types.h File Reference	273
5.18.1	Detailed Description	274
5.19	mainpage.h File Reference	274
6	Example Documentation	275
6.1	examples/main_2darray.c	275
6.2	examples/main_bstree.c	277
6.3	examples/main_hash.c	280
6.4	examples/main_heap.c	284
6.5	examples/main_interval_heap.c	287
6.6	examples/main_list.c	291
6.7	examples/main_llbintree.c	298

6.8	examples/main_llbtree.c	300
6.9	examples/main_lllist.c	302
6.10	examples/main_perm.c	304
6.11	examples/main_queue.c	307
6.12	examples/main_rbtrees.c	310
6.13	examples/main_sort.c	314
6.14	examples/main_stack.c	315

Chapter 1

gdsl

1.1 Introduction

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, search binary trees, red-black trees, 2D arrays, permutations, heaps and interval heaps.

1.2 About

This is the gdsl (Release 1.8) latex-documentation.

1.3 Copyright

Copyright (C) 1998-2018 Nicolas Darnis <ndarnis@free.fr>.

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.3 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the GNU Free Documentation Licence is included in chapter "GNU Free Documentation License".

1.4 Project Manager

Nicolas Darnis <ndarnis@free.fr>.

1.5 Authors

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

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

1.6 Thanks

This is the list of persons (in randomized order) the 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. 236).

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

For his bug report in **gdsl_hash_search()** (p. 112).

- 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. 161), and for the problem of re-defining bool type in **gdsl_types.h** (p. 273).

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

For his gdsl.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 gdsl_interval_heap module.

- Benny Pasternak <bennypk>

For his is bug report in gdsl_rbtrees_map_infix function.

The GDSDL Team.

1.7 GNU Free Documentation License

GNU Free Documentation License Version 1.3, 3 November 2008

Copyright (C) 2000, 2001, 2002, 2007, 2008 Free Software Foundation, Inc. <<https://fsf.org/>> Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

0. PREAMBLE

The purpose of this License is to make a manual, textbook, or other functional and useful document "free" in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondly, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this - License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you". You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not "Transparent" is called "Opaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

The "publisher" means any person or entity that distributes copies of the Document to the public.

A section "Entitled XYZ" means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in

another language. (Here XYZ stands for a specific section name mentioned below, such as "Acknowledgements", "Dedications", "Endorsements", or "History".) To "Preserve the Title" of such a section when you modify the Document means that it remains a section "Entitled XYZ" according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or non-commercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. - If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you

with an updated version of the Document.

4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

A. Use in the Title Page (and on the covers, if any) a title distinct from that of the - Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission. B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement. C. State on the Title page the name of the publisher of the Modified Version, as the publisher. D. Preserve all the copyright notices of the Document. E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices. F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below. G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice. H. Include an unaltered copy of this License. I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence. J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission. K. For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein. L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles. M. Delete any section Entitled "Endorsements". Such a section may not be included in the Modified Version. N. Do not retitle any existing section to be Entitled "Endorsements" or to conflict in title with any Invariant Section. O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section Entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties--for example, statements of

peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled "History" in the various original documents, forming one section Entitled "History"; likewise combine any sections Entitled "Acknowledgements", and any sections Entitled "Dedications". You must delete all sections Entitled "Endorsements".

6. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an "aggregate" if the copyright resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit. When the

Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

8. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

If a section in the Document is Entitled "Acknowledgements", "Dedications", or "History", the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

9. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense, or distribute it is void, and will automatically terminate your rights under this License.

However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, receipt of a copy of some or all of the same material does not give you any rights to use it.

10. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free - Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See <https://www.gnu.org/licenses/>.

Each version of the License is given a distinguishing version number. If the Document

specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation. If the Document specifies that a proxy can decide which future versions of this License can be used, that proxy's public statement of acceptance of a version permanently authorizes you to choose that version for the Document.

11. RELICENSING

"Massive Multiauthor Collaboration Site" (or "MMC Site") means any World Wide Web server that publishes copyrightable works and also provides prominent facilities for anybody to edit those works. A public wiki that anybody can edit is an example of such a server. A "Massive Multiauthor Collaboration" (or "MMC") contained in the site means any set of copyrightable works thus published on the MMC site.

"CC-BY-SA" means the Creative Commons Attribution-Share Alike 3.0 license published by Creative Commons Corporation, a not-for-profit corporation with a principal place of business in San Francisco, California, as well as future copyleft versions of that license published by that same organization.

"Incorporate" means to publish or republish a Document, in whole or in part, as part of another Document.

An MMC is "eligible for relicensing" if it is licensed under this License, and if all works that were first published under this License somewhere other than this MMC, and subsequently incorporated in whole or in part into the MMC, (1) had no cover texts or invariant sections, and (2) were thus incorporated prior to November 1, 2008.

The operator of an MMC Site may republish an MMC contained in the site under CC-BY-SA on the same site at any time before August 1, 2009, provided the MMC is eligible for relicensing.

ADDENDUM: How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

Copyright (c) YEAR YOUR NAME. Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.3 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled "GNU Free Documentation License".

If you have Invariant Sections, Front-Cover Texts and Back-Cover Texts, replace the "with...Texts." line with this:

with the Invariant Sections being LIST THEIR TITLES, with the Front-Cover Texts being LIST, and with the Back-Cover Texts being LIST.

If you have Invariant Sections without Cover Texts, or some other combination of the three, merge those two alternatives to suit the situation.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the

GNU General Public License, to permit their use in free software.

Chapter 2

Module Index

2.1 Modules

Here is a list of all modules:

Low level binary tree manipulation module.	15
Low-level binary search tree manipulation module.	36
Low-level doubly-linked list manipulation module.	53
Low-level doubly-linked node manipulation module.	63
Main module	73
2D-Arrays manipulation module.	74
Binary search tree manipulation module.	84
Hashtable manipulation module.	99
Heap manipulation module.	116
Interval Heap manipulation module.	128
Doubly-linked list manipulation module.	142
Various macros module.	178
Permutation manipulation module.	180
Queue manipulation module.	198
Red-black tree manipulation module.	212
Sort module.	227
Stack manipulation module.	228
GDSDL types.	243

Chapter 3

File Index

3.1 File List

Here is a list of all files with brief descriptions:

_gdsI_bintree.h	Low level binary tree manipulation module	249
_gdsI_bstree.h	Low level binary search tree manipulation module	251
_gdsI_list.h	Low-level doubly-linked list manipulation module	253
_gdsI_node.h	Low-level doubly-linked node manipulation module	254
gdsI.h	255
gdsI_2darray.h	2D-Arrays manipulation module	255
gdsI_bstree.h	Binary search tree manipulation module	257
gdsI_hash.h	Hashtable manipulation module	258
gdsI_heap.h	Heap manipulation module	260
gdsI_interval_heap.h	Interval Heap manipulation module	261
gdsI_list.h	Doubly-linked list manipulation module	263
gdsI_macros.h	Various macros module	266
gdsI_perm.h	Permutation manipulation module	266
gdsI_queue.h	Queue manipulation module	268
gdsI_rbtrees.h	Red-black tree manipulation module	269

gdsi_sort.h	
Sort module	271
gdsi_stack.h	
Stack manipulation module	271
gdsi_types.h	
GDSL types	273
mainpage.h	274

Chapter 4

Module Documentation

4.1 Low level binary tree manipulation module.

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

Typedefs

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

Functions

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Get the height of a low-level binary tree.

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

Get the size of a low-level binary tree.

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

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

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

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

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

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

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

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

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

Left rotate a low-level binary tree.

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

Right rotate a low-level binary tree.

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

Left-right rotate a low-level binary tree.

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

Right-left rotate a low-level binary tree.

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

Parse a low-level binary tree in prefixed order.

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

Parse a low-level binary tree in infix order.

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

Parse a low-level binary tree in postfix order.

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

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

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

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

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

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

4.1.1 Detailed Description

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

4.1.2 Typedef Documentation

4.1.2.1 typedef struct **_gdsl_bintree*** **_gdsl_bintree_t**

GDSDL 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 55 of file `_gdsl_bintree.h`.

4.1.2.2 typedef int(* **_gdsl_bintree_map_func_t**)(const **_gdsl_bintree_t** TREE, void *USER_DATA)

GDSDL 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

GDSDL_MAP_STOP if the mapping must be stopped.
GDSDL_MAP_CONT if the mapping must be continued.

Definition at line 64 of file `_gdsl_bintree.h`.

4.1.2.3 `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 74 of file `_gdsl_bintree.h`.

4.1.3 Function Documentation

4.1.3.1 `_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. 19)

Examples:

`examples/main_llbintree.c`.

4.1.3.2 `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. 18)

Examples:

`examples/main_llbintree.c`.

4.1.3.3 `_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 `_gdsl_bintree_is_empty(T) == TRUE` or in case of insufficient memory.

See also

`_gdsl_bintree_alloc()` (p. 18)

`_gdsl_bintree_free()` (p. 19)

`_gdsl_bintree_is_empty()` (p. 20)

Examples:

`examples/main_llbintree.c`.

4.1.3.4 `bool _gdsl_bintree_is_empty(const _gdsl_bintree_t T)`

Check if a low-level binary tree is empty.

Note

Complexity: $O(1)$

Precondition

nothing.

Parameters

<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. 21)

`_gdsl_bintree_is_root()` (p. 21)

4.1.3.5 `bool _gsdl_bintree_is_leaf(const _gsdl_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 `_gsdl_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

`_gsdl_bintree_is_empty()` (p. 20)

`_gsdl_bintree_is_root()` (p. 21)

4.1.3.6 `bool _gsdl_bintree_is_root(const _gsdl_bintree_t T)`

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

Note

Complexity: $O(1)$

Precondition

T must be a non-empty `_gsdl_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. 20)
`_gdsi_bintree_is_leaf()` (p. 21)

4.1.3.7 `_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. 26)

Examples:

`examples/main_llbintree.c`.

4.1.3.8 `_gdsi_bintree_t _gdsi_bintree_get_parent(const _gdsi_bintree_t T)`

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

Note

Complexity: $O(1)$

Precondition

T must be a non-empty `_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. 21)

`_gdsl_bintree_set_parent()` (p. 27)

4.1.3.9 `_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

`_gdsl_bintree_get_right()` (p. 24)

`_gdsl_bintree_set_left()` (p. 27)

`_gdsl_bintree_set_right()` (p. 28)

4.1.3.10 `_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

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

Returns

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

See also

`_gdsi_bintree_get_left()` (p. 23)

`_gdsi_bintree_set_left()` (p. 27)

`_gdsi_bintree_set_right()` (p. 28)

4.1.3.11 `_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. 25)

4.1.3.12 `_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. 24)

4.1.3.13 `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. 26)

4.1.3.14 `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. 25)

4.1.3.15 `void _gdsi_bintree_set_content(_gdsi_bintree_t T, const gdsi_element_t E)`

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

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

Note

Complexity: $O(1)$

Precondition

T must be a non-empty `_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. 22)

4.1.3.16 `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. 22)

4.1.3.17 `void _gdsl_bintree_set_left(_gdsl_bintree_t T, const _gdsl_bintree_t L)`

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

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

Note

Complexity: $O(1)$

Precondition

T must be a non-empty `_gdsl_bintree_t`.

Parameters

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

See also

`_gdsl_bintree_set_right()` (p. 28)
`_gdsl_bintree_get_left()` (p. 23)
`_gdsl_bintree_get_right()` (p. 24)

4.1.3.18 `void _gdsl_bintree_set_right(_gdsl_bintree_t T, const _gdsl_bintree_t R)`

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

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

Note

Complexity: $O(1)$

Precondition

T must be a non-empty `_gdsl_bintree_t`.

Parameters

<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. 27)
`_gdsl_bintree_get_left()` (p. 23)
`_gdsl_bintree_get_right()` (p. 24)

4.1.3.19 `_gdsl_bintree_t _gdsl_bintree_rotate_left(_gdsl_bintree_t * T)`

Left rotate a low-level binary tree.

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

Note

Complexity: $O(1)$

Precondition

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

Parameters

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

Returns

the modified T left-rotated.

See also

`_gdsI_bintree_rotate_right()` (p. 29)

`_gdsI_bintree_rotate_left_right()` (p. 30)

`_gdsI_bintree_rotate_right_left()` (p. 30)

Examples:

`examples/main_llbintree.c`.

4.1.3.20 `_gdsI_bintree_t _gdsI_bintree_rotate_right(_gdsI_bintree_t * T)`

Right rotate a low-level binary tree.

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

Note

Complexity: $O(1)$

Precondition

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

Parameters

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

Returns

the modified T right-rotated.

See also

`_gdsl_bintree_rotate_left()` (p. 28)
`_gdsl_bintree_rotate_left_right()` (p. 30)
`_gdsl_bintree_rotate_right_left()` (p. 30)

Examples:

`examples/main_llbintree.c.`

4.1.3.21 `_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

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

Returns

the modified T left-right-rotated.

See also

`_gdsl_bintree_rotate_left()` (p. 28)
`_gdsl_bintree_rotate_right()` (p. 29)
`_gdsl_bintree_rotate_right_left()` (p. 30)

4.1.3.22 `_gdsl_bintree_t _gdsl_bintree_rotate_right_left(_gdsl_bintree_t * T)`

Right-left rotate a low-level binary tree.

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

Note

Complexity: $O(1)$

Precondition

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

Parameters

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

Returns

the modified T right-left-rotated.

See also

`_gdsl_bintree_rotate_left()` (p. 28)
`_gdsl_bintree_rotate_right()` (p. 29)
`_gdsl_bintree_rotate_left_right()` (p. 30)

4.1.3.23 `_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 `GD_SL_MAP_STOP`, then `_gdsl_bintree_map_prefix()` (p. 31) stops and returns its last examined node.

Note

Complexity: $O(|T|)$

Precondition

`MAP_F` != NULL

Parameters

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

Returns

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

See also

`_gdsl_bintree_map_infix()` (p. 32)
`_gdsl_bintree_map_postfix()` (p. 33)

Examples:

`examples/main_llbintree.c`.

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

Parse a low-level binary tree in infix order.

Parse all nodes of the low-level binary tree T in infix order. The MAP_F function is called on each node with the USER_DATA argument. If MAP_F returns GDSL_MAP_STOP, then `_gdsl_bintree_map_infix()` (p. 32) 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. 31)
`_gdsl_bintree_map_postfix()` (p. 33)

Examples:

`examples/main_llbintree.c`.

4.1.3.25 `_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. 33) 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. 31)
`_gdsl_bintree_map_infix()` (p. 32)

Examples:

`examples/main_llbintree.c.`

4.1.3.26 `void _gdsl_bintree_write(const _gdsl_bintree_t T, const _gdsl_bintree_write_func_t WRITE_F, FILE * OUTPUT_FILE, void * USER_DATA)`

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

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

Note

Complexity: $O(|T|)$

Precondition

`WRITE_F != NULL & OUTPUT_FILE != NULL`

Parameters

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

See also

`_gdsl_bintree_write_xml()` (p. 34)
`_gdsl_bintree_dump()` (p. 35)

4.1.3.27 `void _gdsl_bintree_write_xml(const _gdsl_bintree_t T, const
_gdsl_bintree_write_func_t WRITE_F, FILE * OUTPUT_FILE, void * USER_DATA)`

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

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

Note

Complexity: $O(|T|)$

Precondition

`OUTPUT_FILE != NULL`

Parameters

<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 <i>WRITE_F</i> .

See also

`_gdsI_bintree_write()` (p. 33)
`_gdsI_bintree_dump()` (p. 35)

Examples:

`examples/main_llbintree.c`.

4.1.3.28 `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. 33)
`_gdsI_bintree_write_xml()` (p. 34)

Examples:

`examples/main_llbintree.c`.

4.2 Low-level binary search tree manipulation module.

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

Typedefs

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

Functions

- **_gdsl_bstree_t _gdsl_bstree_alloc** (const **gdsl_element_t** E)
Create a new low-level binary search tree.
- void **_gdsl_bstree_free** (_gdsl_bstree_t T, const **gdsl_free_func_t** FREE_F)
Destroy a low-level binary search tree.
- **_gdsl_bstree_t _gdsl_bstree_copy** (const _gdsl_bstree_t T, const **gdsl_copy_func_t** COPY_F)
Copy a low-level binary search tree.
- **bool _gdsl_bstree_is_empty** (const _gdsl_bstree_t T)
Check if a low-level binary search tree is empty.
- **bool _gdsl_bstree_is_leaf** (const _gdsl_bstree_t T)
Check if a low-level binary search tree is reduced to a leaf.
- **gdsl_element_t _gdsl_bstree_get_content** (const _gdsl_bstree_t T)
Get the root content of a low-level binary search tree.
- **bool _gdsl_bstree_is_root** (const _gdsl_bstree_t T)
Check if a low-level binary search tree is a root.
- **_gdsl_bstree_t _gdsl_bstree_get_parent** (const _gdsl_bstree_t T)
Get the parent tree of a low-level binary search tree.
- **_gdsl_bstree_t _gdsl_bstree_get_left** (const _gdsl_bstree_t T)
Get the left sub-tree of a low-level binary search tree.
- **_gdsl_bstree_t _gdsl_bstree_get_right** (const _gdsl_bstree_t T)
Get the right sub-tree of a low-level binary search tree.
- **ulong _gdsl_bstree_get_size** (const _gdsl_bstree_t T)
Get the size of a low-level binary search tree.
- **ulong _gdsl_bstree_get_height** (const _gdsl_bstree_t T)
Get the height of a low-level binary search tree.

- **`_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.

4.2.1 Detailed Description

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

4.2.2 Typedef Documentation

4.2.2.1 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 53 of file `_gdsi_bstree.h`.

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

GDSDL 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

GDSDL_MAP_STOP if the mapping must be stopped.
GDSDL_MAP_CONT if the mapping must be continued.

Definition at line 62 of file `_gdsl_bstree.h`.

4.2.2.3 `typedef void(* _gdsl_bstree_write_func_t)(_gdsl_bstree_t TREE, FILE *OUTPUT_FILE, void *USER_DATA)`

GDSDL 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 72 of file `_gdsl_bstree.h`.

4.2.3 Function Documentation

4.2.3.1 `_gdsl_bstree_t _gdsl_bstree_alloc(const gdsl_element_t E)`

Create a new low-level binary search tree.

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

Note

Complexity: $O(1)$

Precondition

nothing.

Parameters

<i>E</i>	The root content of the new low-level binary search tree to create.
----------	---

Returns

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

See also

`_gdsI_bstree_free()` (p. 39)

Examples:

`examples/main_llbstree.c.`

4.2.3.2 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

`_gdsI_bstree_alloc()` (p. 38)

Examples:

`examples/main_llbstree.c.`

4.2.3.3 `_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.

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

Note

Complexity: $O(|T|)$

Precondition

COPY_F != NULL.

Parameters

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

Returns

a copy of T in case of success.

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

See also

`_gdsI_bstree_alloc()` (p. 38)

`_gdsI_bstree_free()` (p. 39)

`_gdsI_bstree_is_empty()` (p. 40)

4.2.3.4 `bool _gdsI_bstree_is_empty(const _gdsI_bstree_t T)`

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

Note

Complexity: $O(1)$

Precondition

nothing.

Parameters

<i>T</i>	The low-level binary search tree to check.
----------	--

Returns

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

See also

`_gdsi_bstree_is_leaf()` (p. 41)
`_gdsi_bstree_is_root()` (p. 42)

4.2.3.5 `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

<i>T</i>	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. 40)
`_gdsi_bstree_is_root()` (p. 42)

4.2.3.6 `_gdsi_element_t _gdsi_bstree_get_content(const _gdsi_bstree_t T)`

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

Note

Complexity: $O(1)$

Precondition

T must be a non-empty `_gdsi_bstree_t`.

Parameters

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

Returns

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

Examples:

examples/main_llbstree.c.

4.2.3.7 `bool _gdsi_bstree_is_root(const _gdsi_bstree_t T)`

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

Note

Complexity: $O(1)$

Precondition

T must be a non-empty `_gdsi_bstree_t`.

Parameters

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

Returns

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

See also

`_gdsi_bstree_is_empty()` (p. 40)
`_gdsi_bstree_is_leaf()` (p. 41)

4.2.3.8 `_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. 42)

4.2.3.9 `_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. 44)

4.2.3.10 `_gdsi_bstree_t _gdsi_bstree_get_right(const _gdsi_bstree_t T)`

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

Note

Complexity: $O(1)$

Precondition

T must be a non-empty `_gdsi_bstree_t`.

Parameters

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

Returns

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

See also

`_gdsi_bstree_get_left()` (p. 43)

4.2.3.11 `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. 45)

4.2.3.12 `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

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

Returns

the height of T .

See also

`_gdsi_bstree_get_size()` (p. 44)

4.2.3.13 `_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

$COMP_F \neq \text{NULL}$ & $RESULT \neq \text{NULL}$.

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. 47)
`_gdsl_bstree_remove()` (p. 46)

Examples:

`examples/main_llbstree.c`.

4.2.3.14 `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. 45)
`_gdsi_bstree_search()` (p. 47)

4.2.3.15 `_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

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

Returns

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

See also

`_gdsi_bstree_insert()` (p. 45)
`_gdsi_bstree_remove()` (p. 46)

4.2.3.16 `_gdsi_bstree_t _gdsi_bstree_search_next(const _gdsi_bstree_t T, const
 gdsi_compare_func_t COMP_F, const gdsi_element_t VALUE)`

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

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

Note

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

Precondition

COMP_F != NULL.

Parameters

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

Returns

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

4.2.3.17 `_gdsI_bstree_t _gdsI_bstree_map_prefix(const _gdsI_bstree_t T, const _gdsI_bstree_map_func_t MAP_F, void * USER_DATA)`

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

Parse all nodes of the low-level binary search tree *T* in prefixed order. The *MAP_F* function is called on each node with the *USER_DATA* argument. If *MAP_F* returns - *GDSL_MAP_STOP*, then `_gdsI_bstree_map_prefix()` (p. 48) 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 <i>MAP_F</i> .

Returns

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

See also

`_gdsl_bstree_map_infix()` (p. 49)
`_gdsl_bstree_map_postfix()` (p. 50)

4.2.3.18 `_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 - GDSL_MAP_STOP, then `_gdsl_bstree_map_infix()` (p. 49) 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. 48)
`_gdsl_bstree_map_postfix()` (p. 50)

4.2.3.19 `_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. 50) 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. 48)
`_gdsl_bstree_map_infix()` (p. 49)

4.2.3.20 `void _gdsl_bstree_write(const _gdsl_bstree_t T, const _gdsl_bstree_write_func_t WRITE_F, FILE * OUTPUT_FILE, void * USER_DATA)`

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

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

Note

Complexity: $O(|T|)$

Precondition

`WRITE_F != NULL & OUTPUT_FILE != NULL.`

Parameters

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

See also

`_gdsi_bstree_write_xml()` (p. 51)

`_gdsi_bstree_dump()` (p. 52)

4.2.3.21 `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 <i>WRITE_F</i> .

See also

`_gdsi_bstree_write()` (p. 50)
`_gdsi_bstree_dump()` (p. 52)

Examples:

`examples/main_llbstree.c`.

4.2.3.22 `void _gdsi_bstree_dump(const _gdsi_bstree_t T, const
_gdsi_bstree_write_func_t WRITE_F, FILE * OUTPUT_FILE, void * USER_DATA)`

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

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

Note

Complexity: $O(|T|)$

Precondition

OUTPUT_FILE != NULL.

Parameters

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

See also

`_gdsi_bstree_write()` (p. 50)
`_gdsi_bstree_write_xml()` (p. 51)

4.3 Low-level doubly-linked list manipulation module.

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

Typedefs

- typedef **_gds_l_node_t _gds_l_list_t**
GDSL low-level doubly-linked list type.

Functions

- **_gds_l_list_t _gds_l_list_alloc** (const **gds_l_element_t** E)
Create a new low-level list.
- void **_gds_l_list_free** (**_gds_l_list_t** L, const **gds_l_free_func_t** FREE_F)
Destroy a low-level list.
- **bool _gds_l_list_is_empty** (const **_gds_l_list_t** L)
Check if a low-level list is empty.
- **ulong _gds_l_list_get_size** (const **_gds_l_list_t** L)
Get the size of a low-level list.
- void **_gds_l_list_link** (**_gds_l_list_t** L1, **_gds_l_list_t** L2)
Link two low-level lists together.
- void **_gds_l_list_insert_after** (**_gds_l_list_t** L, **_gds_l_list_t** PREV)
Insert a low-level list after another one.
- void **_gds_l_list_insert_before** (**_gds_l_list_t** L, **_gds_l_list_t** SUCC)
Insert a low-level list before another one.
- void **_gds_l_list_remove** (**_gds_l_node_t** NODE)
Remove a node from a low-level list.
- **_gds_l_list_t _gds_l_list_search** (**_gds_l_list_t** L, const **gds_l_compare_func_t** COMP_F, void *VALUE)
Search for a particular node in a low-level list.
- **_gds_l_list_t _gds_l_list_map_forward** (const **_gds_l_list_t** L, const **_gds_l_node_map_func_t** MAP_F, void *USER_DATA)
Parse a low-level list in forward order.
- **_gds_l_list_t _gds_l_list_map_backward** (const **_gds_l_list_t** L, const **_gds_l_node_map_func_t** MAP_F, void *USER_DATA)
Parse a low-level list in backward order.
- void **_gds_l_list_write** (const **_gds_l_list_t** L, const **_gds_l_node_write_func_t** WRITE_F, FILE *OUTPUT_FILE, void *USER_DATA)
Write all nodes of a low-level list to a file.
- void **_gds_l_list_write_xml** (const **_gds_l_list_t** L, const **_gds_l_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 **_gds_l_list_dump** (const **_gds_l_list_t** L, const **_gds_l_node_write_func_t** WRITE_F, FILE *OUTPUT_FILE, void *USER_DATA)
Dump the internal structure of a low-level list to a file.

4.3.1 Detailed Description

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

4.3.2 Typedef Documentation

4.3.2.1 typedef `_gdsl_node_t _gdsl_list_t`

GDSDL low-level doubly-linked list type.

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

Definition at line 55 of file `_gdsl_list.h`.

4.3.3 Function Documentation

4.3.3.1 `_gdsl_list_t _gdsl_list_alloc(const gdsl_element_t E)`

Create a new low-level list.

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

Note

Complexity: $O(1)$

Precondition

nothing.

Parameters

<code>E</code>	The content of the first node of the new low-level list to create.
----------------	--

Returns

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

See also

`_gdsl_list_free()` (p. 55)

Examples:

`examples/main_llist.c`.

4.3.3.2 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. 54)

Examples:

examples/main_llist.c.

4.3.3.3 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

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

Returns

TRUE if the low-level list L is empty.

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

4.3.3.4 `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

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

Returns

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

Examples:

examples/main_llist.c.

4.3.3.5 `void _gdsi_list_link(_gdsi_list_t L1, _gdsi_list_t L2)`

Link two low-level lists together.

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

Note

Complexity: $O(|L1|)$

Precondition

L1 & *L2* must be non-empty `_gdsi_list_t`.

Parameters

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

Examples:

examples/main_llist.c.

4.3.3.6 void `_gdsI_list_insert_after`(`_gdsI_list_t L`, `_gdsI_list_t PREV`)

Insert a low-level list after another one.

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

Note

Complexity: $O(|L|)$

Precondition

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

Parameters

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

See also

`_gdsI_list_insert_before`() (p. 57)

`_gdsI_list_remove`() (p. 58)

4.3.3.7 void `_gdsI_list_insert_before`(`_gdsI_list_t L`, `_gdsI_list_t SUCC`)

Insert a low-level list before another one.

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

Note

Complexity: $O(|L|)$

Precondition

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

Parameters

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

See also

`_gdsI_list_insert_after`() (p. 57)

`_gdsI_list_remove`() (p. 58)

4.3.3.8 void `_gdsi_list_remove(_gdsi_node_t NODE)`

Remove a node from a low-level list.

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

Note

Complexity: $O(1)$

Precondition

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

Parameters

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

See also

`_gdsi_list_insert_after()` (p. 57)
`_gdsi_list_insert_before()` (p. 57)

4.3.3.9 `_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.

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

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

Returns

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

4.3.3.10 `_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. 59) 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. 59)

Examples:

`examples/main_llist.c`.

4.3.3.11 `_gdsl_list_t_gdsl_list_map_backward(const _gdsl_list_t L, const
 _gdsl_node_map_func_t MAP_F, void * USER_DATA)`

Parse a low-level list in backward order.

Parse all nodes of the low-level list *L* in backward order. The *MAP_F* function is called on each node with the *USER_DATA* argument. If *MAP_F* returns *GDSL_MAP_STOP*, then ***_gdsl_list_map_backward()*** (p. 59) 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 *GDSL_MAP_STOP*.
NULL when the parsing is done.

See also

_gdsl_list_map_forward() (p. 59)

Examples:

examples/main_llist.c.

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

See also

`_gdsl_list_write_xml()` (p. 61)

`_gdsl_list_dump()` (p. 62)

Examples:

`examples/main_llist.c`.

4.3.3.13 `void _gdsl_list_write_xml (const _gdsl_list_t L, const
_gdsl_node_write_func_t WRITE_F, FILE * OUTPUT_FILE, void * USER_DATA)`

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

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

Note

Complexity: $O(|L|)$

Precondition

OUTPUT_FILE != NULL.

Parameters

<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. 60)
`_gdsi_list_dump()` (p. 62)

Examples:

`examples/main_llist.c`.

4.3.3.14 `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`. Additional `USER_DATA` argument could be passed to `WRITE_F`.

Note

Complexity: $O(|L|)$

Precondition

`OUTPUT_FILE` != `NULL`.

Parameters

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

See also

`_gdsi_list_write()` (p. 60)
`_gdsi_list_write_xml()` (p. 61)

Examples:

`examples/main_llist.c`.

4.4 Low-level doubly-linked node manipulation module.

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

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

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

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

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

4.4.1 Detailed Description

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

4.4.2 Typedef Documentation

4.4.2.1 typedef struct **_gdsl_node*** **_gdsl_node_t**

GDSL low-level doubly linked node type.

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

Definition at line 54 of file **_gdsl_node.h**.

4.4.2.2 typedef int(* **_gdsl_node_map_func_t**)(const **_gdsl_node_t** NODE, void *USER_DATA)

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

Parameters

<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 63 of file **_gdsl_node.h**.

4.4.2.3 typedef void(* **_gdsl_node_write_func_t**)(const **_gdsl_node_t** NODE, FILE *OUTPUT_FILE, void *USER_DATA)

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

Parameters

<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 73 of file `_gdsl_node.h`.

4.4.3 Function Documentation

4.4.3.1 `_gdsl_node_t _gdsl_node_alloc(void)`

Create a new low-level node.

Allocate a new low-level node data structure.

Note

Complexity: $O(1)$

Precondition

nothing.

Returns

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

See also

`_gdsl_node_free()` (p. 65)

4.4.3.2 `gdsl_element_t _gdsl_node_free(_gdsl_node_t NODE)`

Destroy a low-level node.

Deallocate the low-level node NODE.

Note

$O(1)$

Precondition

NODE != NULL

Returns

the content of `NODE` (without modification).

See also

`_gdsl_node_alloc()` (p. 65)

4.4.3.3 `_gdsl_node_t _gdsl_node_get_succ(const _gdsl_node_t NODE)`

Get the successor of a low-level node.

Note

Complexity: $O(1)$

Precondition

`NODE != NULL`

Parameters

<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

`_gdsl_node_get_pred()` (p. 66)
`_gdsl_node_set_succ()` (p. 67)
`_gdsl_node_set_pred()` (p. 68)

4.4.3.4 `_gdsl_node_t _gdsl_node_get_pred(const _gdsl_node_t NODE)`

Get the predecessor of a low-level node.

Note

Complexity: $O(1)$

Precondition

`NODE != NULL`

Parameters

<i>NODE</i>	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. 66)
[`_gdsi_node_set_succ\(\)`](#) (p. 67)
[`_gdsi_node_set_pred\(\)`](#) (p. 68)

4.4.3.5 `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

<i>NODE</i>	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. 68)

Examples:

`examples/main_llist.c.`

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

Set the successor of a low-level node.

Modify the successor 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

[`_gdsl_node_get_succ\(\)`](#) (p. 66)

4.4.3.7 void `_gdsl_node_set_pred`(`_gdsl_node_t` *NODE*, const `_gdsl_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

[`_gdsl_node_get_pred\(\)`](#) (p. 66)

4.4.3.8 void `_gdsl_node_set_content`(`_gdsl_node_t` *NODE*, const `gdsl_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 <code>NODE</code> .

See also

`_gdsi_node_get_content()` (p. 67)

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

See also

`_gdsi_node_unlink()` (p. 69)

4.4.3.10 void _gdsi_node_unlink(_gdsi_node_t *NODE1*, _gdsi_node_t *NODE2*)

Unlink two low-level nodes.

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

Note

Complexity: $O(1)$

Precondition

`NODE1 != NULL & NODE2 != NULL`

Parameters

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

See also

[`_gdsi_node_link\(\)`](#) (p. 69)

4.4.3.11 `void _gdsi_node_write (const _gdsi_node_t NODE, const
_gdsi_node_write_func_t WRITE_F, FILE * OUTPUT_FILE, void * USER_DATA)`

Write a low-level node to a file.

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

Note

Complexity: $O(1)$

Precondition

`NODE != NULL & WRITE_F != NULL & OUTPUT_FILE != NULL`

Parameters

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

See also

`_gdsi_node_write_xml()` (p. 71)

`_gdsi_node_dump()` (p. 71)

4.4.3.12 `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

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

See also

`_gdsi_node_write()` (p. 70)

`_gdsi_node_dump()` (p. 71)

4.4.3.13 `void _gdsi_node_dump(const _gdsi_node_t NODE, const
_gdsi_node_write_func_t WRITE_F, FILE * OUTPUT_FILE, void * USER_DATA)`

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

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

Note

Complexity: $O(1)$

Precondition

`NODE != NULL & OUTPUT_FILE != NULL`

Parameters

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

See also

`_gdsl_node_write()` (p. 70)

`_gdsl_node_write_xml()` (p. 71)

4.5 Main module

GDSL main module.

Functions

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

4.5.1 Detailed Description

GDSL main module.

4.5.2 Function Documentation

4.5.2.1 `const char* gdsl_get_version (void)`

Get GDSL version number as a string.

Note

Complexity: $O(1)$

Precondition

nothing.

Postcondition

the returned string MUST NOT be deallocated.

Returns

the GDSL version number as a string.

4.6 2D-Arrays manipulation module.

This module is for manipulation of 2D-arrays.

Typedefs

- typedef struct gdsI_2darray * **gdsI_2darray_t**
GDSL 2D-array type.

Functions

- **gdsI_2darray_t gdsI_2darray_alloc** (const char *NAME, const **ulong** R, const **ulong** C, const **gdsI_alloc_func_t** ALLOC_F, const **gdsI_free_func_t** FREE_F)
Create a new 2D-array.
- void **gdsI_2darray_free** (**gdsI_2darray_t** A)
Destroy a 2D-array.
- const char * **gdsI_2darray_get_name** (const **gdsI_2darray_t** A)
Get the name of a 2D-array.
- **ulong gdsI_2darray_get_rows_number** (const **gdsI_2darray_t** A)
Get the number of rows of a 2D-array.
- **ulong gdsI_2darray_get_columns_number** (const **gdsI_2darray_t** A)
Get the number of columns of a 2D-array.
- **ulong gdsI_2darray_get_size** (const **gdsI_2darray_t** A)
Get the size of a 2D-array.
- **gdsI_element_t gdsI_2darray_get_content** (const **gdsI_2darray_t** A, const **ulong** R, const **ulong** C)
Get an element from a 2D-array.
- **gdsI_2darray_t gdsI_2darray_set_name** (**gdsI_2darray_t** A, const char *NEW_NAME)
Set the name of a 2D-array.
- **gdsI_element_t gdsI_2darray_set_content** (**gdsI_2darray_t** A, const **ulong** R, const **ulong** C, void *VALUE)
Modify an element in a 2D-array.
- void **gdsI_2darray_write** (const **gdsI_2darray_t** A, const **gdsI_write_func_t** WRITE_F, FILE *OUTPUT_FILE, void *USER_DATA)
Write the content of a 2D-array to a file.
- void **gdsI_2darray_write_xml** (const **gdsI_2darray_t** A, const **gdsI_write_func_t** WRITE_F, FILE *OUTPUT_FILE, void *USER_DATA)
Write the content of a 2D array to a file into XML.
- void **gdsI_2darray_dump** (const **gdsI_2darray_t** A, const **gdsI_write_func_t** WRITE_F, FILE *OUTPUT_FILE, void *USER_DATA)
Dump the internal structure of a 2D array to a file.

4.6.1 Detailed Description

This module is for manipulation of 2D-arrays.

4.6.2 Typedef Documentation

4.6.2.1 typedef struct gdsl_2darray* gdsl_2darray_t

GDSDL 2D-array type.

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

Definition at line 54 of file gdsl_2darray.h.

4.6.3 Function Documentation

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

Create a new 2D-array.

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

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

Note

Complexity: $O(1)$

Precondition

nothing

Parameters

<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. 76)
gdsI_alloc_func_t (p. 244)
gdsI_free_func_t (p. 244)

Examples:

examples/main_2darray.c.

4.6.3.2 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. 75) is used to free elements from A, but no check is done to see if an element was set (ie. != NULL) or not. It's up to you to check if the element to free is NULL or not into the FREE_F function.

Note

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

Precondition

A must be a valid gdsI_2darray_t

Parameters

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

See also

gdsI_2darray_alloc() (p. 75)

Examples:

examples/main_2darray.c.

4.6.3.3 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. 80)

Examples:

`examples/main_2darray.c`.

4.6.3.4 `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. 78)

gdsI_2darray_get_size() (p. 78)

Examples:

examples/main_2darray.c.

4.6.3.5 `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. 77)

gdsI_2darray_get_size() (p. 78)

Examples:

examples/main_2darray.c.

4.6.3.6 `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. 77)

`gdsi_2darray_get_columns_number()` (p. 78)

**4.6.3.7 `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

A	The 2D-array from which getting the element
R	The row index of the element to get
C	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. 80)

**4.6.3.8 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. 76)

**4.6.3.9 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. 79)

Examples:

`examples/main_2darray.c`.

4.6.3.10 `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 A's elements
<i>USER_DATA</i>	User's datas passed to WRITE_F

See also

gdsl_2darray_write_xml() (p. 82)
gdsl_2darray_dump() (p. 82)

Examples:

examples/main_2darray.c.

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

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

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

Note

Complexity: $O(R \times C)$, 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

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

See also

gdsl_2darray_write() (p. 81)
gdsl_2darray_dump() (p. 82)

Examples:

examples/main_2darray.c.

4.6.3.12 `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. Additional USER_DATA argument could be passed to WRITE_F.

Note

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

Precondition

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

Parameters

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

See also

`gdsi_2darray_write()` (p. 81)

`gdsi_2darray_write_xml()` (p. 82)

Examples:

`examples/main_2darray.c`.

4.7 Binary search tree manipulation module.

This module is for manipulation of binary search trees.

Typedefs

- typedef struct gdsi_bstree * **gdsi_bstree_t**
GDSL binary search tree type.

Functions

- **gdsi_bstree_t gdsi_bstree_alloc** (const char *NAME, **gdsi_alloc_func_t** ALL-OC_F, **gdsi_free_func_t** FREE_F, **gdsi_compare_func_t** COMP_F)
Create a new binary search tree.
- void **gdsi_bstree_free** (**gdsi_bstree_t** T)
Destroy a binary search tree.
- void **gdsi_bstree_flush** (**gdsi_bstree_t** T)
Flush a binary search tree.
- const char * **gdsi_bstree_get_name** (const **gdsi_bstree_t** T)
Get the name of a binary search tree.
- bool **gdsi_bstree_is_empty** (const **gdsi_bstree_t** T)
Check if a binary search tree is empty.
- **gdsi_element_t gdsi_bstree_get_root** (const **gdsi_bstree_t** T)
Get the root of a binary search tree.
- **ulong gdsi_bstree_get_size** (const **gdsi_bstree_t** T)
Get the size of a binary search tree.
- **ulong gdsi_bstree_get_height** (const **gdsi_bstree_t** T)
Get the height of a binary search tree.
- **gdsi_bstree_t gdsi_bstree_set_name** (**gdsi_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.

4.7.1 Detailed Description

This module is for manipulation of binary search trees.

4.7.2 Typedef Documentation

4.7.2.1 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 54 of file gdsi_bstree.h.

4.7.3 Function Documentation

4.7.3.1 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. 86)
gdsI_bstree_flush() (p. 87)
gdsI_alloc_func_t (p. 244)
gdsI_free_func_t (p. 244)
gdsI_compare_func_t (p. 245)

Examples:

examples/main_bstree.c.

4.7.3.2 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. 85). 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

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

See also

`gdsl_bstree_alloc()` (p. 85)

`gdsl_bstree_flush()` (p. 87)

Examples:

`examples/main_bstree.c`.

4.7.3.3 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_rbtrees_alloc()`** (p. 213). 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

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

See also

`gdsl_bstree_alloc()` (p. 85)

`gdsl_bstree_free()` (p. 86)

Examples:

`examples/main_bstree.c`.

4.7.3.4 `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. 90) ()

4.7.3.5 `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.

4.7.3.6 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

T	The binary search tree to get the root element from
----------	---

Returns

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

Examples:

examples/main_bstree.c.

4.7.3.7 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

gdsl_bstree_get_height() (p. 90)

Examples:

examples/main_bstree.c.

4.7.3.8 `ulong gsdl_bstree_get_height(const gsdl_bstree_t T)`

Get the height of a binary search tree.

Note

Complexity: $O(|T|)$

Precondition

T must be a valid `gsdl_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

gsdl_bstree_get_size() (p. 89)

Examples:

examples/main_bstree.c.

**4.7.3.9 `gsdl_bstree_t gsdl_bstree_set_name(gsdl_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. 88)

4.7.3.10 `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. 92)

gdsl_bstree_delete() (p. 93)

Examples:

examples/main_bstree.c.

4.7.3.11 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. 85). 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. 91)

gdsl_bstree_delete() (p. 93)

4.7.3.12 `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. 85). 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. 85), 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. 91)
gdsi_bstree_remove() (p. 92)

Examples:

examples/main_bstree.c.

4.7.3.13 `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. 85) is used.

Note

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

Precondition

T must be a valid `gdsl_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

`gdsl_bstree_insert()` (p. 91)
`gdsl_bstree_remove()` (p. 92)
`gdsl_bstree_delete()` (p. 93)

Examples:

`examples/main_bstree.c`.

4.7.3.14 `gdsl_element_t gsdl_bstree_map_prefix(const gsdl_bstree_t T, gsdl_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 **`gsdl_bstree_map_prefix()`** (p. 94) 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_infix() (p. 95)
gdsl_bstree_map_postfix() (p. 96)

4.7.3.15 `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. 95) 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. 94)
gdsl_bstree_map_postfix() (p. 96)

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

Parse a binary search tree in postfix order.

Parse all nodes of the binary search tree T in postfix order. The MAP_F function is called on the element contained in each node with the USER_DATA argument. If MAP_F returns GDSL_MAP_STOP, then **gdsl_bstree_map_postfix()** (p. 96) 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. 94)
gdsl_bstree_map_infix() (p. 95)

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

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

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

Note

Complexity: $O(|T|)$

Precondition

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

Parameters

<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. 97)

`gdsl_bstree_dump()` (p. 98)

Examples:

`examples/main_bstree.c`.

4.7.3.18 `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. 96)
gdsI_bstree_dump() (p. 98)

Examples:

examples/main_bstree.c.

**4.7.3.19 void gdsI_bstree_dump(const gdsI_bstree_t T, gdsI_write_func_t WRITE_F,
FILE * OUTPUT_FILE, void * USER_DATA)**

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

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

Note

Complexity: $O(|T|)$

Precondition

T must be a valid gdsI_bstree_t & OUTPUT_FILE != NULL

Parameters

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

See also

gdsI_bstree_write() (p. 96)
gdsI_bstree_write_xml() (p. 97)

Examples:

examples/main_bstree.c.

4.8 Hashtable manipulation module.

This module is for manipulation of hashtables.

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.

4.8.1 Detailed Description

This module is for manipulation of hashtables.

4.8.2 Typedef Documentation

4.8.2.1 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 54 of file gdsi_hash.h.

4.8.2.2 typedef const char*(* gdsi_key_func_t)(void *VALUE)

GDSL hashtable key function type.

Postcondition

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

Parameters

VALUE	The value used to get the key from
--------------	------------------------------------

Returns

The key associated to the VALUE.

Definition at line 62 of file gdsI_hash.h.

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

GDSL hashtable hash function type.

Parameters

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

Returns

The hashed value computed from KEY.

Definition at line 70 of file gdsI_hash.h.

4.8.3 Function Documentation**4.8.3.1 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.

4.8.3.2 `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. 111) 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. 101) 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. 103)
gdsI_hash_flush() (p. 103)
gdsI_hash_insert() (p. 108)
gdsI_hash_modify() (p. 111)

Examples:

examples/main_hash.c.

4.8.3.3 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. 102). 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. 102)
gdsI_hash_flush() (p. 103)

Examples:

examples/main_hash.c.

4.8.3.4 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. 102). 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. 102)

`gdsI_hash_free()` (p. 103)

Examples:

`examples/main_hash.c`.

4.8.3.5 `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. 108)

4.8.3.6 `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. 106)
`gdsI_hash_fill_factor()`

4.8.3.7 `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. 111) 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. 105)
gdsl_hash_get_longest_list_size() (p. 106)
gdsl_hash_modify() (p. 111)

4.8.3.8 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. 106)
gdsl_hash_fill_factor()
gdsl_hash_get_entries_number() (p. 105)
gdsl_hash_get_lists_max_size() (p. 105)

4.8.3.9 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. 105)

`gdsI_hash_fill_factor()`

`gdsI_hash_get_longest_list_size()` (p. 106)

4.8.3.10 `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. 105)

`gdsI_hash_get_longest_list_size()` (p. 106)

`gdsI_hash_get_size()` (p. 106)

Examples:

examples/main_hash.c.

4.8.3.11 `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. 104)

4.8.3.12 `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. 102).

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. 102)
`gdsI_hash_remove()` (p. 109)
`gdsI_hash_delete()` (p. 110)
`gdsI_hash_get_size()` (p. 106)
`gdsI_hash_get_entries_number()` (p. 105)
`gdsI_hash_modify()` (p. 111)

Examples:

`examples/main_hash.c`.

4.8.3.13 `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. 108)
gdsI_hash_search() (p. 112)
gdsI_hash_delete() (p. 110)

Examples:

examples/main_hash.c.

4.8.3.14 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. 102), 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. 108)

gdsI_hash_search() (p. 112)

gdsI_hash_remove() (p. 109)

4.8.3.15 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. 108).

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. 108)

gdsI_hash_get_entries_number() (p. 105)

gdsI_hash_get_fill_factor() (p. 107)

gdsI_hash_get_longest_list_size() (p. 106)

gdsI_hash_get_lists_max_size() (p. 105)

4.8.3.16 `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. 108)

`gdsI_hash_remove()` (p. 109)

`gdsI_hash_delete()` (p. 110)

Examples:

`examples/main_hash.c`.

4.8.3.17 `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. 112) 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.

4.8.3.18 `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. 114)

`gdsl_hash_dump()` (p. 114)

Examples:

`examples/main_hash.c`.

4.8.3.19 `void gdsi_hash_write_xml(const gdsi_hash_t H, gdsi_write_func_t WRITE_F, FILE * OUTPUT_FILE, void * USER_DATA)`

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

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

Note

Complexity: $O(|H|)$

Precondition

H must be a valid gdsi_hash_t & OUTPUT_FILE != NULL

Parameters

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

See also

gdsi_hash_write() (p. 113)
gdsi_hash_dump() (p. 114)

Examples:

examples/main_hash.c.

4.8.3.20 `void gdsi_hash_dump(const gdsi_hash_t H, gdsi_write_func_t WRITE_F, FILE * OUTPUT_FILE, void * USER_DATA)`

Dump the internal structure of a hashtable to a file.

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

Note

Complexity: $O(|H|)$

Precondition

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

Parameters

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

See also

`gdsi_hash_write()` (p. 113)

`gdsi_hash_write_xml()` (p. 114)

Examples:

`examples/main_hash.c`.

4.9 Heap manipulation module.

This module is for manipulation of heaps.

Typedefs

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

Functions

- **gdsl_heap_t** **gdsl_heap_alloc** (const char *NAME, **gdsl_alloc_func_t** ALLOC_F, **gdsl_free_func_t** FREE_F, **gdsl_compare_func_t** COMP_F)
Create a new heap.
- void **gdsl_heap_free** (**gdsl_heap_t** H)
Destroy a heap.
- void **gdsl_heap_flush** (**gdsl_heap_t** H)
Flush a heap.
- const char * **gdsl_heap_get_name** (const **gdsl_heap_t** H)
Get the name of a heap.
- **ulong** **gdsl_heap_get_size** (const **gdsl_heap_t** H)
Get the size of a heap.
- **gdsl_element_t** **gdsl_heap_get_top** (const **gdsl_heap_t** H)
Get the top of a heap.
- **bool** **gdsl_heap_is_empty** (const **gdsl_heap_t** H)
Check if a heap is empty.
- **gdsl_heap_t** **gdsl_heap_set_name** (**gdsl_heap_t** H, const char *NEW_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 **gdsI_heap_write_xml** (const **gdsI_heap_t** H, **gdsI_write_func_t** WRITE_F, FILE *OUTPUT_FILE, void *USER_DATA)
Write the content of a heap to a file into XML.
- void **gdsI_heap_dump** (const **gdsI_heap_t** H, **gdsI_write_func_t** WRITE_F, FILE *OUTPUT_FILE, void *USER_DATA)
Dump the internal structure of a heap to a file.

4.9.1 Detailed Description

This module is for manipulation of heaps.

4.9.2 Typedef Documentation

4.9.2.1 typedef struct heap* **gdsI_heap_t**

GDSL heap type.

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

Definition at line 55 of file gdsI_heap.h.

4.9.3 Function Documentation

4.9.3.1 **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. 118)
gdsI_heap_flush() (p. 119)

Examples:

examples/main_heap.c.

4.9.3.2 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. 117). 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. 117)
gdsI_heap_flush() (p. 119)

Examples:

examples/main_heap.c.

4.9.3.3 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. 117). *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. 117)

`gdsI_heap_free()` (p. 118)

Examples:

`examples/main_heap.c`.

4.9.3.4 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. 121)

Examples:

examples/main_heap.c.

4.9.3.5 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 $|H|$).

4.9.3.6 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. 122)

Examples:

examples/main_heap.c.

4.9.3.7 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.

4.9.3.8 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 `gdsl_heap_t`

Parameters

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

Returns

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

See also

`gdsl_heap_get_name()` (p. 119)

4.9.3.9 `gdsl_element_t gdsl_heap_set_top(gdsl_heap_t H, void * VALUE)`

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

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

Note

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

Precondition

H must be a valid `gdsl_heap_t`

Parameters

<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. 120)

Examples:

examples/main_heap.c.

4.9.3.10 **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. 117)

gdsI_heap_remove()

gdsI_heap_delete()

gdsI_heap_get_size() (p. 120)

Examples:

examples/main_heap.c.

4.9.3.11 **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. 123)

`gdsI_heap_delete_top()` (p. 124)

4.9.3.12 `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. 117), 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. 123)
gdsI_heap_remove_top() (p. 123)

Examples:

examples/main_heap.c.

4.9.3.13 **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 <i>MAP_F</i>

Returns

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

Examples:

examples/main_heap.c.

4.9.3.14 **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 `gdsl_heap_t` & `OUTPUT_FILE` != NULL & `WRITE_F` != NULL

Parameters

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

See also

`gdsl_heap_write_xml()` (p. 126)

`gdsl_heap_dump()` (p. 127)

4.9.3.15 `void gdsl_heap_write_xml(const gdsl_heap_t H, gdsl_write_func_t WRITE_F, FILE * OUTPUT_FILE, void * USER_DATA)`

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

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

Note

Complexity: $O(|H|)$

Precondition

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

Parameters

<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 <code>WRITE_F</code> .

See also

gdsI_heap_write() (p. 125)

gdsI_heap_dump() (p. 127)

Examples:

examples/main_heap.c.

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

See also

gdsI_heap_write() (p. 125)

gdsI_heap_write_xml() (p. 126)

Examples:

examples/main_heap.c.

4.10 Interval Heap manipulation module.

This module is for manipulation of interval heaps.

Typedefs

- typedef struct heap * **gdsl_interval_heap_t**

GDSL interval heap type.

Functions

- **gdsl_interval_heap_t** **gdsl_interval_heap_alloc** (const char *NAME, **gdsl_alloc_func_t** ALLOC_F, **gdsl_free_func_t** FREE_F, **gdsl_compare_func_t** COMP_F)

Create a new interval heap.

- void **gdsl_interval_heap_free** (**gdsl_interval_heap_t** H)

Destroy an interval heap.

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

Flush an interval heap.

- const char * **gdsl_interval_heap_get_name** (const **gdsl_interval_heap_t** H)

Get the name of an interval heap.

- **ulong** **gdsl_interval_heap_get_size** (const **gdsl_interval_heap_t** H)

Get the size of a interval heap.

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

Set the maximum size of the interval heap.

- **bool** **gdsl_interval_heap_is_empty** (const **gdsl_interval_heap_t** H)

Check if an interval heap is empty.

- **gdsl_interval_heap_t** **gdsl_interval_heap_set_name** (**gdsl_interval_heap_t** H, const char *NEW_NAME)

Set the name of an interval heap.

- **gdsl_element_t** **gdsl_interval_heap_insert** (**gdsl_interval_heap_t** H, void *VALUE)

Insert an element into an interval heap (PUSH).

- **gdsl_element_t** **gdsl_interval_heap_remove_max** (**gdsl_interval_heap_t** H)

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

- **gdsl_element_t** **gdsl_interval_heap_remove_min** (**gdsl_interval_heap_t** H)

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

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

Get the minimum element.

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

Get the maximum element.

- **gdsI_interval_heap_t gdsI_interval_heap_delete_min** (**gdsI_interval_heap_t** H)

Delete the minimum element from an interval heap.

- **gdsI_interval_heap_t gdsI_interval_heap_delete_max** (**gdsI_interval_heap_t** H)

Delete the maximum element from an interval heap.

- **gdsI_element_t gdsI_interval_heap_map_forward** (const **gdsI_interval_heap_t** H, **gdsI_map_func_t** MAP_F, void *USER_DATA)

Parse a interval heap.

- void **gdsI_interval_heap_write** (const **gdsI_interval_heap_t** H, **gdsI_write_func_t** WRITE_F, FILE *OUTPUT_FILE, void *USER_DATA)

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

- 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.

- 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.

4.10.1 Detailed Description

This module is for manipulation of interval heaps.

4.10.2 Typedef Documentation

4.10.2.1 typedef struct heap* gdsI_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 54 of file gdsI_interval_heap.h.

4.10.3 Function Documentation

4.10.3.1 **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

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

Returns

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

See also

gdsI_interval_heap_free() (p. 130)
gdsI_interval_heap_flush() (p. 131)

Examples:

examples/main_interval_heap.c.

4.10.3.2 void gdsI_interval_heap_free(gdsI_interval_heap_t H)

Destroy an interval heap.

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

Note

Complexity: $O(|H|)$

Precondition

H must be a valid `gdsl_interval_heap_t`

Parameters

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

See also

`gdsl_interval_heap_alloc()` (p. 129)

`gdsl_interval_heap_flush()` (p. 131)

Examples:

`examples/main_interval_heap.c`.

4.10.3.3 `void gdsl_interval_heap_flush(gdsl_interval_heap_t H)`

Flush an interval heap.

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

Note

Complexity: $O(|H|)$

Precondition

H must be a valid `gdsl_interval_heap_t`

Parameters

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

See also

`gdsl_interval_heap_alloc()` (p. 129)

`gdsl_interval_heap_free()` (p. 130)

Examples:

`examples/main_interval_heap.c`.

4.10.3.4 `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. 134)

4.10.3.5 `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 get the size from
----------	--

Returns

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

Examples:

examples/main_interval_heap.c.

**4.10.3.6 void gdsI_interval_heap_set_max_size(const gdsI_interval_heap_t H ,
ulong $size$)**

Set the maximum size of the interval heap.

Note

Complexity: $O(1)$

Precondition

H must be a valid gdsI_interval_heap_t

Parameters

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

Returns

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

4.10.3.7 bool gdsI_interval_heap_is_empty(const gdsI_interval_heap_t H)

Check if an interval heap is empty.

Note

Complexity: $O(1)$

Precondition

H must be a valid gdsI_interval_heap_t

Parameters

H	The interval heap to check
-----	----------------------------

Returns

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

4.10.3.8 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. 132)

4.10.3.9 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 `gdsI_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

`gdsI_interval_heap_alloc()` (p. 129)
`gdsI_interval_heap_remove()`
`gdsI_interval_heap_delete()`
`gdsI_interval_heap_get_size()` (p. 132)

Examples:

`examples/main_interval_heap.c.`

4.10.3.10 `gdsI_element_t gdsI_interval_heap_remove_max (gdsI_interval_heap_t H)`

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

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

Note

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

Precondition

H must be a valid `gdsI_interval_heap_t`

Parameters

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

Returns

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

See also

gdsl_interval_heap_insert() (p. 134)
gdsl_interval_heap_delete_max() (p. 138)

Examples:

examples/main_interval_heap.c.

4.10.3.11 gdsl_element_t gdsl_interval_heap_remove_min(gdsl_interval_heap_t H)

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

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

Note

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

Precondition

H must be a valid gdsl_interval_heap_t

Parameters

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

Returns

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

See also

gdsl_interval_heap_insert() (p. 134)
gdsl_interval_heap_delete_max() (p. 138)

Examples:

examples/main_interval_heap.c.

4.10.3.12 `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

4.10.3.13 `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

4.10.3.14 `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. 129), 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. 134)
`gdsi_interval_heap_remove_top()`

4.10.3.15 `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. 129), 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. 134)
 gsdl_interval_heap_remove_top()

4.10.3.16 `gsdl_element_t gsdl_interval_heap_map_forward (const
 gsdl_interval_heap_t H, gsdl_map_func_t MAP_F, void * USER_DATA)`

Parse a interval heap.

Parse all elements of the interval heap H. The MAP_F function is called on each H's element with USER_DATA argument. If MAP_F returns GDSL_MAP_STOP then gsdl_interval_heap_map() stops and returns its last examined element.

Note

Complexity: $O(|H|)$

Precondition

H must be a valid gsdl_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 MAP_F

Returns

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

4.10.3.17 `void gsdl_interval_heap_write (const gsdl_interval_heap_t H,
 gsdl_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 H's elements.
<i>USER_DATA</i>	User's datas passed to <code>WRITE_F</code> .

See also

`gdsI_interval_heap_write_xml()` (p. 140)
`gdsI_interval_heap_dump()` (p. 141)

4.10.3.18 `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`.
Additional `USER_DATA` argument could be passed to `WRITE_F`.

Note

Complexity: $O(|H|)$

Precondition

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

Parameters

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

See also

gdsI_interval_heap_write() (p. 139)

gdsI_interval_heap_dump() (p. 141)

4.10.3.19 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*. 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. 139)

gdsI_interval_heap_write_xml() (p. 140)

4.11 Doubly-linked list manipulation module.

This module is for manipulation of doubly-linked lists.

Typedefs

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

Functions

- **gdsl_list_t gdsl_list_alloc** (const char *NAME, **gdsl_alloc_func_t** ALLOC_F, **gdsl_free_func_t** FREE_F)
Create a new list.
- void **gdsl_list_free** (**gdsl_list_t** L)
Destroy a list.
- void **gdsl_list_flush** (**gdsl_list_t** L)
Flush a list.
- const char * **gdsl_list_get_name** (const **gdsl_list_t** L)
Get the name of a list.
- **ulong gdsl_list_get_size** (const **gdsl_list_t** L)
Get the size of a list.
- **bool gdsl_list_is_empty** (const **gdsl_list_t** L)
Check if a list is empty.
- **gdsl_element_t gdsl_list_get_head** (const **gdsl_list_t** L)
Get the head of a list.
- **gdsl_element_t gdsl_list_get_tail** (const **gdsl_list_t** L)
Get the tail of a list.
- **gdsl_list_t gdsl_list_set_name** (**gdsl_list_t** L, const char *NEW_NAME)
Set the name of a list.
- **gdsl_element_t gdsl_list_insert_head** (**gdsl_list_t** L, void *VALUE)
Insert an element at the head of a list.
- **gdsl_element_t gdsl_list_insert_tail** (**gdsl_list_t** L, void *VALUE)
Insert an element at the tail of a list.
- **gdsl_element_t gdsl_list_remove_head** (**gdsl_list_t** L)
Remove the head of a list.
- **gdsl_element_t gdsl_list_remove_tail** (**gdsl_list_t** L)
Remove the tail of a list.
- **gdsl_element_t gdsl_list_remove** (**gdsl_list_t** L, **gdsl_compare_func_t** COMP_F, const void *VALUE)
Remove a particular element from a list.

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

- **gdsl_element_t** **gdsl_list_cursor_move_to_value** (**gdsl_list_cursor_t** C, **gdsl_compare_func_t** COMP_F, void *VALUE)
Place a cursor on a particular element.
- **gdsl_element_t** **gdsl_list_cursor_move_to_position** (**gdsl_list_cursor_t** C, **ulong** POS)
Place a cursor on a element given by its position.
- void **gdsl_list_cursor_step_forward** (**gdsl_list_cursor_t** C)
Move a cursor one step forward of its list.
- void **gdsl_list_cursor_step_backward** (**gdsl_list_cursor_t** C)
Move a cursor one step backward of its list.
- **bool** **gdsl_list_cursor_is_on_head** (const **gdsl_list_cursor_t** C)
Check if a cursor is on the head of its list.
- **bool** **gdsl_list_cursor_is_on_tail** (const **gdsl_list_cursor_t** C)
Check if a cursor is on the tail of its list.
- **bool** **gdsl_list_cursor_has_succ** (const **gdsl_list_cursor_t** C)
Check if a cursor has a successor.
- **bool** **gdsl_list_cursor_has_pred** (const **gdsl_list_cursor_t** C)
Check if a cursor has a predecessor.
- void **gdsl_list_cursor_set_content** (**gdsl_list_cursor_t** C, **gdsl_element_t** E)
Set the content of the cursor.
- **gdsl_element_t** **gdsl_list_cursor_get_content** (const **gdsl_list_cursor_t** C)
Get the content of a cursor.
- **gdsl_element_t** **gdsl_list_cursor_insert_after** (**gdsl_list_cursor_t** C, void *VALUE)
Insert a new element after a cursor.
- **gdsl_element_t** **gdsl_list_cursor_insert_before** (**gdsl_list_cursor_t** C, void *VALUE)
Insert a new element before a cursor.
- **gdsl_element_t** **gdsl_list_cursor_remove** (**gdsl_list_cursor_t** C)
Remove the element under a cursor.
- **gdsl_element_t** **gdsl_list_cursor_remove_after** (**gdsl_list_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.

4.11.1 Detailed Description

This module is for manipulation of doubly-linked lists.

4.11.2 Typedef Documentation

4.11.2.1 `typedef struct _gdsl_list* gsdl_list_t`

GDSL doubly-linked list type.

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

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

4.11.2.2 `typedef struct _gsdl_list_cursor* gsdl_list_cursor_t`

GDSL doubly-linked list cursor type.

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

Definition at line 60 of file `gsdl_list.h`.

4.11.3 Function Documentation

4.11.3.1 `gsdl_list_t gsdl_list_alloc (const char * NAME, gsdl_alloc_func_t ALLOC_F, gsdl_free_func_t FREE_F)`

Create a new list.

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

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

Note

Complexity: $O(1)$

Precondition

nothing

Parameters

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

Returns

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

See also

gdsl_list_free() (p. 146)
gdsl_list_flush() (p. 147)

Examples:

examples/main_list.c.

4.11.3.2 void gsdl_list_free (gsdl_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 **gsdl_list_alloc()** (p. 145).

Note

Complexity: $O(|L|)$

Precondition

L must be a valid gsdl_list_t

Parameters

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

See also

gsdl_list_alloc() (p. 145)
gsdl_list_flush() (p. 147)

Examples:

examples/main_list.c.

4.11.3.3 void `gdsl_list_flush`(`gdsl_list_t` *L*)

Flush a list.

Destroy all the elements of the list *L* by calling *L*'s `FREE_F` function passed to **`gdsl_list_alloc()`** (p. 145). *L* is not deallocated itself and *L*'s name is not modified.

Note

Complexity: $O(|L|)$

Precondition

L must be a valid `gdsl_list_t`

Parameters

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

See also

`gdsl_list_alloc()` (p. 145)

`gdsl_list_free()` (p. 146)

Examples:

`examples/main_list.c`.

4.11.3.4 const char* `gdsl_list_get_name`(const `gdsl_list_t` *L*)

Get the name of a list.

Note

Complexity: $O(1)$

Precondition

L must be a valid `gdsl_list_t`

Postcondition

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

Parameters

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

Returns

the name of the list `L`.

See also

`gdsi_list_set_name()` (p. 150)

Examples:

`examples/main_list.c`.

4.11.3.5 `ulong gdsi_list_get_size(const gdsi_list_t L)`

Get the size of a list.

Note

Complexity: $O(1)$

Precondition

`L` must be a valid `gdsi_list_t`

Parameters

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

Returns

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

Examples:

`examples/main_list.c`.

4.11.3.6 `bool gdsi_list_is_empty(const gdsi_list_t L)`

Check if a list is empty.

Note

Complexity: $O(1)$

Precondition

`L` must be a valid `gdsi_list_t`

Parameters

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

Returns

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

Examples:

examples/main_list.c.

4.11.3.7 `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

<code>L</code>	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. 149)

4.11.3.8 `gdsi_element_t gdsi_list_get_tail(const gdsi_list_t L)`

Get the tail of a list.

Note

Complexity: $O(1)$

Precondition

L must be a valid `gdsI_list_t`

Parameters

<i>L</i>	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

`gdsI_list_get_head()` (p. 149)

4.11.3.9 `gdsI_list_t gdsI_list_set_name(gdsI_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 `gdsI_list_t`

Parameters

<i>L</i>	The list to change the name
<i>NEW_NAME</i>	The new name of L

Returns

the modified list in case of success.
NULL in case of failure.

See also

`gdsI_list_get_name()` (p. 147)

4.11.3.10 `gdsI_element_t gdsI_list_insert_head(gdsI_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 **`gdsI_list_alloc()`** (p. 145). 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 `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_tail()` (p. 151)
`gdsI_list_remove_head()` (p. 152)
`gdsI_list_remove_tail()` (p. 153)
`gdsI_list_remove()` (p. 153)

Examples:

`examples/main_list.c`.

4.11.3.11 `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. 145). 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. 151)
`gdsi_list_remove_head()` (p. 152)
`gdsi_list_remove_tail()` (p. 153)
`gdsi_list_remove()` (p. 153)

Examples:

`examples/main_list.c`.

4.11.3.12 `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

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

Returns

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

See also

gdsl_list_insert_head() (p. 151)

gdsl_list_insert_tail() (p. 151)

gdsl_list_remove_tail() (p. 153)

gdsl_list_remove() (p. 153)

4.11.3.13 **gdsl_element_t** **gdsl_list_remove_tail**(**gdsl_list_t** *L*)

Remove the tail of a list.

Remove the element at the tail of the list *L*.

Note

Complexity: $O(1)$

Precondition

L must be a valid **gdsl_list_t**

Parameters

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

Returns

the removed element in case of success.

NULL in case of *L* is empty.

See also

gdsl_list_insert_head() (p. 151)

gdsl_list_insert_tail() (p. 151)

gdsl_list_remove_head() (p. 152)

gdsl_list_remove() (p. 153)

4.11.3.14 **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 `gdsi_list_t` & `COMP_F` != NULL

Parameters

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

Returns

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

See also

`gdsi_list_insert_head()` (p. 151)
`gdsi_list_insert_tail()` (p. 151)
`gdsi_list_remove_head()` (p. 152)
`gdsi_list_remove_tail()` (p. 153)

4.11.3.15 `gdsi_list_t gdsi_list_delete_head(gdsi_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 **`gdsi_list_alloc()`** (p. 145).

Note

Complexity: $O(1)$

Precondition

L must be a valid `gdsi_list_t`

Parameters

<i>L</i>	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. 145)
gdsl_list_destroy_tail()
gdsl_list_destroy()

Examples:

examples/main_list.c.

4.11.3.16 **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. 145).

Note

Complexity: $O(1)$

Precondition

L must be a valid **gdsl_list_t**

Parameters

<i>L</i>	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. 145)
gdsl_list_destroy_head()
gdsl_list_destroy()

Examples:

examples/main_list.c.

4.11.3.17 **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 **gdsI_list_alloc()** (p. 145).

Note

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

Precondition

L must be a valid *gdsI_list_t* & *COMP_F* != NULL

Parameters

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

Returns

the modified list *L* if the element is found.
 NULL if the element to destroy is not found.

See also

gdsI_list_alloc() (p. 145)
gdsI_list_destroy_head()
gdsI_list_destroy_tail()

Examples:

examples/main_list.c.

**4.11.3.18 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 L's element with VALUE
<i>VALUE</i>	The value to compare L'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. 157)
gdsI_list_search_max() (p. 158)
gdsI_list_search_min() (p. 159)

Examples:

examples/main_list.c.

4.11.3.19 **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. 156)
gdsi_list_search_max() (p. 158)
gdsi_list_search_min() (p. 159)

Examples:

examples/main_list.c.

4.11.3.20 **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* != NULL

Parameters

<i>L</i>	The list to search the element in
<i>COMP_F</i>	The comparison function to use to compare <i>L</i> '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. 156)
gdsi_list_search_by_position() (p. 157)
gdsi_list_search_min() (p. 159)

Examples:

examples/main_list.c.

4.11.3.21 `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

<code>L</code>	The list to search the element in
<code>COMP_F</code>	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. 156)
`gdsI_list_search_by_position()` (p. 157)
`gdsI_list_search_max()` (p. 158)

4.11.3.22 `gdsI_list_t gdsI_list_sort (gdsI_list_t L, gdsI_compare_func_t COMP_F)`

Sort a list.

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

Note

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

Precondition

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

Parameters

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

Returns

the sorted list L.

Examples:

examples/main_list.c.

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

Parse a list from head to tail.

Parse all elements of the list L from head to tail. The MAP_F function is called on each L's element with USER_DATA argument. If MAP_F returns GDSL_MAP_STOP, then **gdsl_list_map_forward()** (p. 160) stops and returns its last examined element.

Note

Complexity: $O(|L|)$

Precondition

L must be a valid gdsl_list_t & MAP_F != NULL

Parameters

<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_backward() (p. 161)

4.11.3.24 **gdsI_element_t gdsI_list_map_backward(const gdsI_list_t L,
gdsI_map_func_t MAP_F, void * USER_DATA)**

Parse a list from tail to head.

Parse all elements of the list L from tail to head. The MAP_F function is called on each L's element with USER_DATA argument. If MAP_F returns GDSL_MAP_STOP then **gdsI_list_map_backward()** (p. 161) stops and returns its last examined element.

Note

Complexity: $O(|L|)$

Precondition

L must be a valid gdsI_list_t & MAP_F != NULL

Parameters

<i>L</i>	The list to parse
<i>MAP_F</i>	The map function to apply on each 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

gdsI_list_map_forward() (p. 160)

Examples:

examples/main_list.c.

4.11.3.25 **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 `gdsl_list_t` & `OUTPUT_FILE` != NULL & `WRITE_F` != NULL

Parameters

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

See also

`gdsl_list_write_xml()` (p. 162)

`gdsl_list_dump()` (p. 163)

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

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

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

Note

Complexity: $O(|L|)$

Precondition

L must be a valid `gdsl_list_t` & `OUTPUT_FILE` != NULL

Parameters

<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 <code>WRITE_F</code> .

See also

`gdsl_list_write()` (p. 161)

`gdsl_list_dump()` (p. 163)

Examples:

examples/main_list.c.

4.11.3.27 `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

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

See also

`gdsI_list_write()` (p. 161)

`gdsI_list_write_xml()` (p. 162)

Examples:

examples/main_list.c.

4.11.3.28 `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 `gdsl_list_t`

Parameters

L	The list on wich the cursor is positionned.
----------	---

Returns

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

See also

`gdsl_list_cursor_free()` (p. 164)

Examples:

`examples/main_list.c`.

4.11.3.29 void `gdsl_list_cursor_free(gdsl_list_cursor_t C)`

Destroy a list cursor.

Note

Complexity: $O(1)$

Precondition

C must be a valid `gdsl_list_cursor_t`.

Parameters

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

See also

`gdsl_list_cursor_alloc()` (p. 163)

Examples:

`examples/main_list.c`.

4.11.3.30 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. 165)

Examples:

examples/main_list.c.

4.11.3.31 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. 165)

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

Place a cursor on a particular element.

Search a particular element E in the cursor's list L by comparing all list's elements to VALUE, by using COMP_F. If E is found, C is positionned on it.

Note

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

Precondition

C must be a valid `gdsl_list_cursor_t` & `COMP_F` != NULL

Parameters

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

Returns

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

See also

`gdsl_list_cursor_move_to_position()` (p. 166)

Examples:

`examples/main_list.c`.

4.11.3.33 `gdsl_element_t gdsl_list_cursor_move_to_position(gdsl_list_cursor_t C, ulong POS)`

Place a cursor on a element given by its position.

Search for the POS-th element in the cursor's list L. In case this element exists, the cursor C is positionned on it.

Note

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

Precondition

C must be a valid `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. 166)

4.11.3.34 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 `gdsi_list_cursor_t`

Parameters

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

See also

`gdsi_list_cursor_step_backward()` (p. 167)

Examples:

`examples/main_list.c`.

4.11.3.35 void `gdsi_list_cursor_step_backward(gdsi_list_cursor_t C)`

Move a cursor one step backward of its list.

Move the cursor `C` one node backward (from tail to head.) Does nothing if `C` is already on its list's head.

Note

Complexity: $O(1)$

Precondition

`C` must be a valid `gdsl_list_cursor_t`

Parameters

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

See also

`gdsl_list_cursor_step_forward()` (p. 167)

Examples:

`examples/main_list.c`.

4.11.3.36 `bool gdsl_list_cursor_is_on_head(const gdsl_list_cursor_t C)`

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

Note

Complexity: $O(1)$

Precondition

`C` must be a valid `gdsl_list_cursor_t`

Parameters

<code>C</code>	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. 169)

4.11.3.37 bool gdsI_list_cursor_is_on_tail(const gdsI_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 gdsI_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

gdsI_list_cursor_is_on_head() (p. 168)

4.11.3.38 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. 170)

4.11.3.39 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. 169)

4.11.3.40 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. 171) 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. 171)

4.11.3.41 **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. 170)

Examples:

examples/main_list.c.

4.11.3.42 **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. 145). If the returned value is not NULL, then the new element is placed after the cursor *C*. If *C*'s list is empty, the element is inserted at the head position of *C*'s list.

Note

Complexity: $O(1)$

Precondition

C must be a valid `gdsl_list_cursor_t`

Parameters

C	The cursor after which the new element must be inserted
VALUE	The value used to allocate the new element to insert

Returns

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

See also

`gdsl_list_cursor_insert_before()` (p. 172)
`gdsl_list_cursor_remove_after()` (p. 174)
`gdsl_list_cursor_remove_before()` (p. 174)

Examples:

`examples/main_list.c`.

4.11.3.43 `gdsl_element_t` `gdsl_list_cursor_insert_before` (`gdsl_list_cursor_t` C, void * *VALUE*)

Insert a new element before a cursor.

A new element is created using `ALLOC_F` called on `VALUE`. `ALLOC_F` is the pointer passed to **`gdsl_list_alloc()`** (p. 145). If the returned value is not NULL, then the new element is placed before the cursor C. If C's list is empty, the element is inserted at the head position of C's list.

Note

Complexity: $O(1)$

Precondition

C must be a valid `gdsl_list_cursor_t`

Parameters

C	The cursor before which the new element must be inserted
VALUE	The value used to allocate the new element to insert

Returns

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

See also

gdsI_list_cursor_insert_after() (p. 171)
gdsI_list_cursor_remove_after() (p. 174)
gdsI_list_cursor_remove_before() (p. 174)

Examples:

examples/main_list.c.

4.11.3.44 gdsI_element_t gdsI_list_cursor_remove(gdsI_list_cursor_t C)

Removec the element under a cursor.

Note

Complexity: $O(1)$

Precondition

C must be a valid gdsI_list_cursor_t

Postcondition

After this operation, the cursor is positionned on to its successor.

Parameters

C	The cursor to remove the content from.
---	--

Returns

the removed element if it exists.
NULL if there is not element to remove.

See also

gdsI_list_cursor_insert_after() (p. 171)
gdsI_list_cursor_insert_before() (p. 172)
gdsI_list_cursor_remove() (p. 173)
gdsI_list_cursor_remove_before() (p. 174)

4.11.3.45 `gdsl_element_t` `gdsl_list_cursor_remove_after`(`gdsl_list_cursor_t` *C*)

Removec the element after a cursor.

Note

Complexity: $O(1)$

Precondition

C must be a valid `gdsl_list_cursor_t`

Parameters

<i>C</i>	The cursor to remove the successor from.
----------	--

Returns

the removed element if it exists.
NULL if there is not element to remove.

See also

`gdsl_list_cursor_insert_after()` (p. 171)
`gdsl_list_cursor_insert_before()` (p. 172)
`gdsl_list_cursor_remove()` (p. 173)
`gdsl_list_cursor_remove_before()` (p. 174)

4.11.3.46 `gdsl_element_t` `gdsl_list_cursor_remove_before`(`gdsl_list_cursor_t` *C*)

Remove the element before a cursor.

Note

Complexity: $O(1)$

Precondition

C must be a valid `gdsl_list_cursor_t`

Parameters

<i>C</i>	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. 171)
gdsI_list_cursor_insert_before() (p. 172)
gdsI_list_cursor_remove() (p. 173)
gdsI_list_cursor_remove_after() (p. 174)

4.11.3.47 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. 145).

Complexity: O(1)

Precondition

C must be a valid gdsI_list_cursor_t

Parameters

C	The cursor to delete the content.
---	-----------------------------------

Returns

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

See also

gdsI_list_cursor_delete_before() (p. 176)
gdsI_list_cursor_delete_after() (p. 175)

4.11.3.48 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. 145).

Complexity: O(1)

Precondition

C must be a valid `gdsl_list_cursor_t`

Parameters

C	The cursor to delete the successor from.
---	--

Returns

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

See also

`gdsl_list_cursor_delete()` (p. 175)
`gdsl_list_cursor_delete_before()` (p. 176)

Examples:

`examples/main_list.c`.

4.11.3.49 `gdsl_list_cursor_t` **`gdsl_list_cursor_delete_before(gdsl_list_cursor_t C)`**

Delete the element before the cursor of a list.

Remove the element before the cursor C. The removed element is also deallocated using `FREE_F` passed to **`gdsl_list_alloc()`** (p. 145).

Note

Complexity: $O(1)$

Precondition

C must be a valid `gdsl_list_cursor_t`

Parameters

C	The cursor to delete the predecessor from.
---	--

Returns

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

See also

gdsI_list_cursor_delete() (p. 175)

gdsI_list_cursor_delete_after() (p. 175)

4.12 Various macros module.

This module provides some various macros.

Defines

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

4.12.1 Detailed Description

This module provides some various macros.

4.12.2 Define Documentation

4.12.2.1 `#define GDSL_MAX(X, Y) (X>Y?X:Y)`

Give the greatest number of two numbers.

Note

Complexity: $O(1)$

Precondition

X & Y must be basic scalar C types

Parameters

X	First scalar variable
Y	Second scalar variable

Returns

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

See also

GDSL_MIN() (p. 179)

Definition at line 57 of file `gdsl_macros.h`.

4.12.2.2 `#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. 178)

Definition at line 74 of file gdsl_macros.h.

4.13 Permutation manipulation module.

This module is for manipulation of permutations.

Typedefs

- typedef struct gdsI_perm * **gdsI_perm_t**
GDSL permutation type.
- typedef void(* **gdsI_perm_write_func_t**)(ulong E, FILE *OUTPUT_FILE, **gdsI-_location_t** POSITION, void *USER_DATA)
GDSL permutation write function type.
- typedef struct gdsI_perm_data * **gdsI_perm_data_t**

Enumerations

- enum **gdsI_perm_position_t**{ **GDSL_PERM_POSITION_FIRST** = 1, **GDSL_P-ERM_POSITION_LAST** = 2 }
- This type is for gdsI_perm_write_func_t.*

Functions

- **gdsI_perm_t gdsI_perm_alloc** (const char *NAME, const ulong N)
Create a new permutation.
- void **gdsI_perm_free** (**gdsI_perm_t** P)
Destroy a permutation.
- **gdsI_perm_t gdsI_perm_copy** (const **gdsI_perm_t** P)
Copy a permutation.
- const char * **gdsI_perm_get_name** (const **gdsI_perm_t** P)
Get the name of a permutation.
- **ulong gdsI_perm_get_size** (const **gdsI_perm_t** P)
Get the size of a permutation.
- **ulong gdsI_perm_get_element** (const **gdsI_perm_t** P, const ulong INDIX)
Get the (INDIX+1)-th element from a permutation.
- **ulong * gdsI_perm_get_elements_array** (const **gdsI_perm_t** P)
Get the array elements of a permutation.
- **ulong gdsI_perm_linear_inversions_count** (const **gdsI_perm_t** P)
Count the inversions number into a linear permutation.
- **ulong gdsI_perm_linear_cycles_count** (const **gdsI_perm_t** P)
Count the cycles number into a linear permutation.
- **ulong gdsI_perm_canonical_cycles_count** (const **gdsI_perm_t** P)
Count the cycles number into a canonical permutation.
- **gdsI_perm_t gdsI_perm_set_name** (**gdsI_perm_t** P, const char *NEW_NAME)

Set the name of a permutation.

- **gdsI_perm_t gdsI_perm_linear_next (gdsI_perm_t P)**

Get the next permutation from a linear permutation.

- **gdsI_perm_t gdsI_perm_linear_prev (gdsI_perm_t P)**

Get the previous permutation from a linear permutation.

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

Initialize a permutation with an array of values.

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

Multiply two permutations.

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

Convert a linear permutation to its canonical form.

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

Convert a canonical permutation to its linear form.

- **gdsI_perm_t gdsI_perm_inverse (gdsI_perm_t P)**

Inverse in place a permutation.

- **gdsI_perm_t gdsI_perm_reverse (gdsI_perm_t P)**

Reverse in place a permutation.

- **gdsI_perm_t gdsI_perm_randomize (gdsI_perm_t P)**

Randomize a permutation.

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

Apply a permutation on to a vector.

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

Write the elements of a permutation to a file.

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

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

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

Dump the internal structure of a permutation to a file.

4.13.1 Detailed Description

This module is for manipulation of permutations.

4.13.2 Typedef Documentation

4.13.2.1 typedef struct gdsl_perm* gdsl_perm_t

GDSL permutation type.

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

Definition at line 51 of file gdsl_perm.h.

4.13.2.2 typedef void(* gdsl_perm_write_func_t)(ulong E, FILE *OUTPUT_FILE, gdsl_location_t POSITION, void *USER_DATA)

GDSL permutation write function type.

Parameters

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

Definition at line 75 of file gdsl_perm.h.

4.13.2.3 typedef struct gdsl_perm_data* gdsl_perm_data_t

Definition at line 81 of file gdsl_perm.h.

4.13.3 Enumeration Type Documentation

4.13.3.1 enum gdsl_perm_position_t

This type is for `gdsl_perm_write_func_t`.

Enumerator:

GDSL_PERM_POSITION_FIRST When element is at first position

GDSL_PERM_POSITION_LAST When element is at last position

Definition at line 56 of file gdsl_perm.h.

4.13.4 Function Documentation

4.13.4.1 gdsI_perm_t gdsI_perm_alloc (const char * NAME, const ulong N)

Create a new permutation.

Allocate a new permutation data structure of size N wich name is set to a copy of NAME.

Note

Complexity: $O(N)$

Precondition

$N > 0$

Parameters

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

Returns

the newly allocated identity permutation in its linear form in case of success.
NULL in case of insufficient memory.

See also

gdsI_perm_free() (p. 183)
gdsI_perm_copy() (p. 184)

Examples:

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

4.13.4.2 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

gdsl_perm_alloc() (p. 183)

gdsl_perm_copy() (p. 184)

Examples:

examples/main_bstree.c, **examples/main_list.c**, **examples/main_llbtree.c**,
examples/main_perm.c, and **examples/main_rbtrees.c**.

4.13.4.3 **gdsl_perm_t** **gdsl_perm_copy** (**const** **gdsl_perm_t** P)

Copy a permutation.

Create and return a copy of the permutation P .

Note

Complexity: $O(|P|)$

Precondition

P must be a valid **gdsl_perm_t**.

Postcondition

The returned permutation must be deallocated with **gdsl_perm_free**.

Parameters

P	The permutation to copy.
-----	--------------------------

Returns

a copy of P in case of success.

NULL in case of insufficient memory.

See also

gdsl_perm_alloc (p. 183)

gdsl_perm_free (p. 183)

4.13.4.4 `const char* gdsl_perm_get_name(const gdsl_perm_t P)`

Get the name of a permutation.

Note

Complexity: $O(1)$

Precondition

P must be a valid `gdsl_perm_t`

Postcondition

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

Parameters

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

Returns

the name of the permutation P.

See also

`gdsl_perm_set_name()` (p. 189)

4.13.4.5 `ulong gdsl_perm_get_size(const gdsl_perm_t P)`

Get the size of a permutation.

Note

Complexity: $O(1)$

Precondition

P must be a valid `gdsl_perm_t`

Parameters

<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. 186)
gdsI_perm_get_elements_array() (p. 186)

4.13.4.6 `ulong gdsI_perm_get_element(const gdsI_perm_t P, const ulong INDIX)`

Get the $(INDIX+1)$ -th element from a permutation.

Note

Complexity: $O(1)$

Precondition

P must be a valid `gdsI_perm_t` & $0 \leq INDIX < |P|$

Parameters

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

Returns

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

See also

gdsI_perm_get_size() (p. 185)
gdsI_perm_get_elements_array() (p. 186)

Examples:

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

4.13.4.7 `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. 186)

`gdsi_perm_set_elements_array()` (p. 191)

4.13.4.8 `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`.

4.13.4.9 `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. 188)

Examples:

`examples/main_perm.c`.

4.13.4.10 `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. 187)

Examples:

examples/main_perm.c.

**4.13.4.11 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
<i>NEW_NAME</i>	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. 185)

4.13.4.12 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. 190)

4.13.4.13 `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. 189)

4.13.4.14 `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. 186)

4.13.4.15 `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: $O(|RESULT|)$

Precondition

RESULT, ALPHA and BETA must be valids `gdsI_perm_t` & $|RESULT| == |ALPHA|$
 $== |BETA|$

Parameters

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

Returns

RESULT, the result of the multiplication ALPHA x BETA.

4.13.4.16 `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. 192)

Examples:

`examples/main_perm.c`.

4.13.4.17 `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. 192)

Examples:

`examples/main_perm.c`.

4.13.4.18 `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. 194)

Examples:

examples/main_perm.c.

4.13.4.19 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

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

Returns

the mirror image of the permutation P

See also

gdsI_perm_inverse() (p. 193)

Examples:

examples/main_perm.c.

4.13.4.20 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_llbtree.c`,
`examples/main_perm.c`, and `examples/main_rbtrees.c`.

4.13.4.21 `gdsl_element_t* gsdl_perm_apply_on_array(gsdl_element_t* V, const gsdl_perm_t P)`

Apply a permutation on to a vector.

Note

Complexity: $O(|P|)$

Precondition

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

Parameters

<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`.

4.13.4.22 `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. Additionnal 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. 196)

`gdsi_perm_dump()` (p. 197)

Examples:

`examples/main_perm.c`.

4.13.4.23 `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. 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 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. 196)

`gdsI_perm_dump()` (p. 197)

4.13.4.24 `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. 196)

`gdsI_perm_write_xml()` (p. 196)

4.14 Queue manipulation module.

This module is for manipulation of queues.

Typedefs

- typedef struct _gdsl_queue * **gdsl_queue_t**
GDSL queue type.

Functions

- **gdsl_queue_t gdsl_queue_alloc** (const char *NAME, **gdsl_alloc_func_t** ALL-OC_F, **gdsl_free_func_t** FREE_F)
Create a new queue.
- void **gdsl_queue_free** (**gdsl_queue_t** Q)
Destroy a queue.
- void **gdsl_queue_flush** (**gdsl_queue_t** Q)
Flush a queue.
- const char * **gdsl_queue_get_name** (const **gdsl_queue_t** Q)
Get the name of a queue.
- **ulong gdsl_queue_get_size** (const **gdsl_queue_t** Q)
Get the size of a queue.
- **bool gdsl_queue_is_empty** (const **gdsl_queue_t** Q)
Check if a queue is empty.
- **gdsl_element_t gdsl_queue_get_head** (const **gdsl_queue_t** Q)
Get the head of a queue.
- **gdsl_element_t gdsl_queue_get_tail** (const **gdsl_queue_t** Q)
Get the tail of a queue.
- **gdsl_queue_t gdsl_queue_set_name** (**gdsl_queue_t** Q, const char *NEW_NAME)
Set the name of a queue.
- **gdsl_element_t gdsl_queue_insert** (**gdsl_queue_t** Q, void *VALUE)
Insert an element in a queue (PUT).
- **gdsl_element_t gdsl_queue_remove** (**gdsl_queue_t** Q)
Remove an element from a queue (GET).
- **gdsl_element_t gdsl_queue_search** (const **gdsl_queue_t** Q, **gdsl_compare_func_t** COMP_F, void *VALUE)
Search for a particular element in a queue.
- **gdsl_element_t gdsl_queue_search_by_position** (const **gdsl_queue_t** Q, **ulong** POS)
Search for an element by its position in a queue.
- **gdsl_element_t gdsl_queue_map_forward** (const **gdsl_queue_t** Q, **gdsl_map_func_t** MAP_F, void *USER_DATA)

Parse a queue from head to tail.

- **gdsl_element_t 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.

4.14.1 Detailed Description

This module is for manipulation of queues.

4.14.2 Typedef Documentation

4.14.2.1 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 55 of file gsdl_queue.h.

4.14.3 Function Documentation

4.14.3.1 **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. 200)
gdsI_queue_flush() (p. 201)

Examples:

examples/main_queue.c.

4.14.3.2 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. 199). 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

<i>Q</i>	The queue to destroy
----------	----------------------

See also

gdsI_queue_alloc() (p. 199)
gdsI_queue_flush() (p. 201)

Examples:

examples/main_queue.c.

4.14.3.3 void **gdsI_queue_flush**(gdsI_queue_t Q)

Flush a queue.

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

Note

Complexity: $O(|Q|)$

Precondition

Q must be a valid **gdsI_queue_t**

Parameters

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

See also

gdsI_queue_alloc() (p. 199)

gdsI_queue_free() (p. 200)

Examples:

examples/main_queue.c.

4.14.3.4 const char* **gdsI_queue_get_name**(const gdsI_queue_t Q)

Getsthe name of a queue.

Note

Complexity: $O(1)$

Precondition

Q must be a valid **gdsI_queue_t**

Postcondition

The returned string MUST NOT be freed.

Parameters

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

Returns

the name of the queue Q.

See also

gdsI_queue_set_name() (p. 204)

Examples:

examples/main_queue.c.

4.14.3.5 `ulong gdsI_queue_get_size(const gdsI_queue_t Q)`

Get the size of a queue.

Note

Complexity: $O(1)$

Precondition

Q must be a valid `gdsI_queue_t`

Parameters

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

Returns

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

4.14.3.6 `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.

4.14.3.7 `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. 204)

Examples:

examples/main_queue.c.

4.14.3.8 `gdsl_element_t gsdl_queue_get_tail(const gsdl_queue_t Q)`

Get the tail of a queue.

Note

Complexity: $O(1)$

Precondition

Q must be a valid `gsdl_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

`gsdl_queue_get_head()` (p. 203)

Examples:

`examples/main_queue.c`.

4.14.3.9 `gsdl_queue_t gsdl_queue_set_name(gsdl_queue_t Q, const char * NEW_NAME)`

Set the name of a queue.

Change the previous name of the queue Q to a copy of NEW_NAME.

Note

Complexity: $O(1)$

Precondition

Q must be a valid `gsdl_queue_t`

Parameters

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

Returns

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

See also

gdsl_queue_get_name() (p. 201)

4.14.3.10 `gsdl_element_t gsdl_queue_insert(gsdl_queue_t Q, void * VALUE)`

Insert an element in a queue (PUT).

Allocate a new element E by calling Q's ALLOC_F function on VALUE. ALLOC_F is the function pointer passed to **gsdl_queue_alloc()** (p. 199). The new element E is then inserted at the header position of the queue Q.

Note

Complexity: $O(1)$

Precondition

Q must be a valid `gsdl_queue_t`

Parameters

Q	The queue to insert in
VALUE	The value used to make the new element to insert into Q

Returns

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

See also

gsdl_queue_remove() (p. 205)

Examples:

examples/main_queue.c.

4.14.3.11 `gsdl_element_t gsdl_queue_remove(gsdl_queue_t Q)`

Remove an element from a queue (GET).

Remove the element at the footer position of the queue Q.

Note

Complexity: $O(1)$

Precondition

Q must be a valid `gdsl_queue_t`

Parameters

Q	The queue to remove the tail from
---	-----------------------------------

Returns

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

See also

`gdsl_queue_insert()` (p. 205)

Examples:

`examples/main_queue.c`.

4.14.3.12 `gdsl_element_t gdsl_queue_search (const gdsl_queue_t Q, gdsl_compare_func_t COMP_F, void * VALUE)`

Search for a particular element in a queue.

Search for the first element E equal to VALUE in the queue Q, by using COMP_F to compare all Q's element with.

Note

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

Precondition

Q must be a valid `gdsl_queue_t` & `COMP_F` != NULL

Parameters

Q	The queue to search the element in
COMP_F	The comparison function used to compare Q's element with VALUE
VALUE	The value to compare Q's elements with

Returns

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

See also

gdsI_queue_search_by_position (p. 207)

4.14.3.13 **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
POS	The position where is the element to search

Returns

the element at the POS-th position in the queue Q.
NULL if $POS > |L|$ or $POS \leq 0$.

See also

gdsI_queue_search() (p. 206)

Examples:

examples/main_queue.c.

4.14.3.14 **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. 207) stops and returns its last examined element.

Note

Complexity: $O(|Q|)$

Precondition

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

Parameters

Q	The queue to parse
MAP_F	The map function to apply on each Q 's element
$USER_DATA$	User's datas passed to MAP_F

Returns

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

See also

`gdsl_queue_map_backward()` (p. 208)

Examples:

`examples/main_queue.c`.

4.14.3.15 `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 `GD_SL_MAP_STOP`, then **`gdsl_queue_map_backward()`** (p. 208) stops and returns its last examined element.

Note

Complexity: $O(|Q|)$

Precondition

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

Parameters

<i>Q</i>	The queue to parse
<i>MAP_F</i>	The map function to apply on each Q'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. Returns NULL when the parsing is done.

See also

gdsl_queue_map_forward() (p. 207)

4.14.3.16 `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 WRITE_F.

See also

gdsl_queue_write_xml() (p. 209)

gdsl_queue_dump() (p. 210)

4.14.3.17 `void gdsl_queue_write_xml(const gdsl_queue_t Q, gdsl_write_func_t
WRITE_F, FILE * OUTPUT_FILE, void * USER_DATA)`

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

Write the elements of the queue `Q` to `OUTPUT_FILE`, into XML language. If `WRITE_F` \neq `NULL`, then uses `WRITE_F` to write `Q`'s elements to `OUTPUT_FILE`. Additional `USER_DATA` argument could be passed to `WRITE_F`.

Note

Complexity: $O(|Q|)$

Precondition

`Q` must be a valid `gdsl_queue_t` & `OUTPUT_FILE` \neq `NULL`

Parameters

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

See also

`gdsl_queue_write()` (p. 209)
`gdsl_queue_dump()` (p. 210)

Examples:

`examples/main_queue.c`.

4.14.3.18 `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.

Dump the structure of the queue `Q` to `OUTPUT_FILE`. If `WRITE_F` \neq `NULL`, then uses `WRITE_F` to write `Q`'s elements to `OUTPUT_FILE`. Additional `USER_DATA` argument could be passed to `WRITE_F`.

Note

Complexity: $O(|Q|)$

Precondition

`Q` must be a valid `gdsl_queue_t` & `OUTPUT_FILE` \neq `NULL`

Parameters

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

See also

gdsi_queue_write() (p. 209)

gdsi_queue_write_xml() (p. 209)

Examples:

examples/main_queue.c.

4.15 Red-black tree manipulation module.

This module is for manipulation of red-black trees.

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.

- **gdsI_element_t gdsI_rbtrees_map_infix** (const **gdsI_rbtrees_t** T, **gdsI_map_func_t** MAP_F, void *USER_DATA)
Parse a red-black tree in infix order.
- **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.
- 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.
- 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.
- 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.

4.15.1 Detailed Description

This module is for manipulation of red-black trees.

4.15.2 Typedef Documentation

4.15.2.1 typedef struct gdsI_rbtrees* gdsI_rbtrees_t

GDSL red-black tree type.

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

Definition at line 53 of file gdsI_rbtrees.h.

4.15.3 Function Documentation

4.15.3.1 **gdsI_rbtrees_t gdsI_rbtrees_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 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. 214)
gdsI_rbtrees_flush() (p. 215)

Examples:

examples/main_rbtrees.c.

4.15.3.2 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. 213). 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

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

See also

gdsI_rbtree_alloc() (p. 213)

gdsI_rbtree_flush() (p. 215)

Examples:

examples/main_rbtree.c.

4.15.3.3 void gdsI_rbtree_flush(gdsI_rbtree_t T)

Flush a red-black tree.

Deallocate all the elements of the red-black tree T by calling T's FREE_F function passed to **gdsI_rbtree_alloc()** (p. 213). The red-black tree T is not deallocated itself and its name is not modified.

Note

Complexity: $O(|T|)$

Precondition

T must be a valid gdsI_rbtree_t

See also

gdsI_rbtree_alloc() (p. 213)

gdsI_rbtree_free() (p. 214)

Examples:

examples/main_rbtree.c.

4.15.3.4 char* gdsI_rbtree_get_name(const gdsI_rbtree_t T)

Get the name of a red-black tree.

Note

Complexity: $O(1)$

Precondition

T must be a valid gdsI_rbtree_t

Postcondition

The returned string MUST NOT be freed.

Parameters

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

Returns

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

See also

gdsI_rbtree_set_name() (p. 218)

4.15.3.5 bool gdsI_rbtree_is_empty(const gdsI_rbtree_t *T*)

Check if a red-black tree is empty.

Note

Complexity: $O(1)$

Precondition

T must be a valid gdsI_rbtree_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_rbtree.c.

4.15.3.6 gdsI_element_t gdsI_rbtree_get_root(const gdsI_rbtree_t *T*)

Get the root of a red-black tree.

Note

Complexity: $O(1)$

Precondition

T must be a valid `gdsl_rbtrees_t`

Parameters

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

Returns

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

Examples:

examples/main_rbtrees.c.

4.15.3.7 `ulong gsdl_rbtrees_get_size(const gsdl_rbtrees_t T)`

Get the size of a red-black tree.

Note

Complexity: $O(1)$

Precondition

T must be a valid `gsdl_rbtrees_t`

Parameters

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

Returns

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

See also

`gsdl_rbtrees_get_height()`

Examples:

examples/main_rbtrees.c.

4.15.3.8 `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

<i>T</i>	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. 217)

Examples:

`examples/main_rbtrees.c`.

4.15.3.9 `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_rbtree_get_name() (p. 215)

4.15.3.10 **gdsI_element_t gdsI_rbtree_insert(gdsI_rbtree_t T, void * VALUE, int * RESULT)**

Insert an element into a red-black tree if it's not found or return it.

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

Note

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

Precondition

T must be a valid gdsI_rbtree_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

gdsI_rbtree_remove() (p. 220)
gdsI_rbtree_delete() (p. 220)

Examples:

examples/main_rbtree.c.

4.15.3.11 `gdsi_element_t gdsi_rbtrees_remove(gdsi_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 **gdsi_rbtrees_alloc()** (p. 213). If E is found, it is removed from T and then returned.

Note

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

Precondition

T must be a valid `gdsi_rbtrees_t`

Parameters

<i>T</i>	The red-black tree to modify
<i>VALUE</i>	The value used to find the element to remove

Returns

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

See also

gdsi_rbtrees_insert() (p. 219)
gdsi_rbtrees_delete() (p. 220)

4.15.3.12 `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. 213). 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. 213), 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_rbtree_insert() (p. 219)
gdsI_rbtree_remove() (p. 220)

Examples:

examples/main_rbtree.c.

4.15.3.13 **gdsI_element_t gdsI_rbtree_search (const gdsI_rbtree_t T,
 gdsI_compare_func_t COMP_F, void * VALUE)**

Search for a particular element into a red-black tree.

Search the first element E equal to VALUE in the red-black tree T, by using COMP_F function to find it. If COMP_F == NULL, then the COMP_F function passed to **gdsI_rbtree_alloc()** (p. 213) is used.

Note

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

Precondition

T must be a valid gdsI_rbtree_t

Parameters

<i>T</i>	The red-black tree to use.
<i>COMP_F</i>	The comparison function to use to compare T's element with VALUE to find the element E (or NULL to use the default T's COMP_F)
<i>VALUE</i>	The value that must be used by COMP_F to find the element E

Returns

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

See also

gdsi_rbtrees_insert() (p. 219)
gdsi_rbtrees_remove() (p. 220)
gdsi_rbtrees_delete() (p. 220)

Examples:

examples/main_rbtrees.c.

4.15.3.14 **gdsi_element_t gdsi_rbtrees_map_prefix(const gdsi_rbtrees_t T, gdsi_map_func_t MAP_F, void * USER_DATA)**

Parse a red-black tree in prefixed order.

Parse all nodes of the red-black tree T in prefixed order. The MAP_F function is called on the element contained in each node with the USER_DATA argument. If MAP_F returns GDSL_MAP_STOP, then **gdsi_rbtrees_map_prefix()** (p. 222) 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_infix() (p. 223)
gdsi_rbtrees_map_postfix() (p. 223)

Examples:

examples/main_rbtrees.c.

4.15.3.15 **gdsI_element_t gdsI_rbtree_map_infix**(const **gdsI_rbtree_t** *T*,
gdsI_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 **gdsI_rbtree_map_infix()** (p. 223) stops and returns its last examined element.

Note

Complexity: $O(|T|)$

Precondition

T must be a valid **gdsI_rbtree_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 <i>MAP_F</i>

Returns

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

See also

gdsI_rbtree_map_prefix() (p. 222)
gdsI_rbtree_map_postfix() (p. 223)

Examples:

examples/main_rbtree.c.

4.15.3.16 **gdsI_element_t gdsI_rbtree_map_postfix**(const **gdsI_rbtree_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_rbtree_map_postfix()** (p. 223) stops and returns its last examined element.

Note

Complexity: $O(|T|)$

Precondition

T must be a valid `gdsl_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 <code>MAP_F</code>

Returns

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

See also

`gdsl_rbtrees_map_prefix()` (p. 222)

`gdsl_rbtrees_map_infix()` (p. 223)

Examples:

`examples/main_rbtrees.c`.

4.15.3.17 `void gdsl_rbtrees_write(const gdsl_rbtrees_t T, gdsl_write_func_t WRITE_F,
FILE * OUTPUT_FILE, void * USER_DATA)`

Write the element of each node of a red-black tree to a file.

Write the nodes elements of the red-black tree T to `OUTPUT_FILE`, using `WRITE_F` function. Additionnal `USER_DATA` argument could be passed to `WRITE_F`.

Note

Complexity: $O(|T|)$

Precondition

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

Parameters

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

See also

gdsI_rbtrees_write_xml() (p. 225)

gdsI_rbtrees_dump() (p. 226)

Examples:

examples/main_rbtrees.c.

4.15.3.18 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.
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. 224)

gdsI_rbtrees_dump() (p. 226)

Examples:

examples/main_rbtrees.c.

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

See also

gdsi_rbtrees_write() (p. 224)

gdsi_rbtrees_write_xml() (p. 225)

Examples:

examples/main_rbtrees.c.

4.16 Sort module.

This module is for sorting arrays.

Functions

- void **gdsi_sort** (**gdsi_element_t** **T*, **ulong** *N*, **const gdsi_compare_func_t** *COMP_F*)
Sort an array in place.

4.16.1 Detailed Description

This module is for sorting arrays.

4.16.2 Function Documentation

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

Sort an array in place.

Sort the array *T* in place. The function *COMP_F* is used to compare *T*'s elements and must be user-defined.

Note

Complexity: $O(N \log(N))$

Precondition

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

Parameters

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

Examples:

examples/main_sort.c.

4.17 Stack manipulation module.

This module is for manipulation of stacks.

Typedefs

- typedef struct `_gdsl_stack` * **gdsl_stack_t**
GDSL stack type.

Functions

- **gdsl_stack_t gdsl_stack_alloc** (const char *NAME, **gdsl_alloc_func_t** 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.

4.17.1 Detailed Description

This module is for manipulation of stacks.

4.17.2 Typedef Documentation

4.17.2.1 typedef struct _gdsl_stack* **gdsl_stack_t**

GDSL stack type.

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

Definition at line 54 of file `gdsl_stack.h`.

4.17.3 Function Documentation

4.17.3.1 **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

`gdsI_stack_free()` (p. 230)
`gdsI_stack_flush()` (p. 231)

Examples:

`examples/main_stack.c`.

4.17.3.2 void `gdsI_stack_free(gdsI_stack_t S)`

Destroy a stack.

Deallocate all the elements of the stack `S` by calling `S`'s `FREE_F` function passed to **`gdsI_stack_alloc()`** (p. 229). The name of `S` is deallocated and `S` is deallocated itself too.

Note

Complexity: $O(|S|)$

Precondition

`S` must be a valid `gdsI_stack_t`

Parameters

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

See also

gdsI_stack_alloc() (p. 229)

gdsI_stack_flush() (p. 231)

Examples:

examples/main_stack.c.

4.17.3.3 void gdsI_stack_flush(gdsI_stack_t S)

Flush a stack.

Deallocate all the elements of the stack S by calling S's FREE_F function passed to **gdsI_stack_alloc()** (p. 229). S is not deallocated itself and S's name is not modified.

Note

Complexity: $O(|S|)$

Precondition

S must be a valid gdsI_stack_t

Parameters

S	The stack to flush
---	--------------------

See also

gdsI_stack_alloc() (p. 229)

gdsI_stack_free() (p. 230)

Examples:

examples/main_stack.c.

4.17.3.4 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. 235)

Examples:

`examples/main_stack.c`.

4.17.3.5 `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 $|S|$).

4.17.3.6 `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. 236).

Note

Complexity: $O(1)$

Precondition

S must be a valid `gdsI_stack_t`

Parameters

<i>S</i>	The stack to get the growing factor from
----------	--

Returns

the growing factor of the stack *S*.

See also

gdsI_stack_insert() (p. 236)

gdsI_stack_set_growing_factor() (p. 236)

4.17.3.7 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

<i>S</i>	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.

4.17.3.8 gdsI_element_t gdsI_stack_get_top(const gdsI_stack_t S)

Get the top of a stack.

Note

Complexity: $O(1)$

Precondition

S must be a valid gdsI_stack_t

Parameters

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

Returns

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

See also

gdsI_stack_get_bottom() (p. 234)

Examples:

examples/main_stack.c.

4.17.3.9 gdsI_element_t gdsI_stack_get_bottom(const gdsI_stack_t S)

Get the bottom of a stack.

Note

Complexity: $O(1)$

Precondition

S must be a valid `gdsI_stack_t`

Parameters

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

Returns

the element contained at the bottom position of the stack S if S is not empty. The returned element is not removed from S.
NULL if the stack S is empty.

See also

`gdsI_stack_get_top()` (p. 234)

**4.17.3.10 `gdsI_stack_t gdsI_stack_set_name (gdsI_stack_t S, const char *
NEW_NAME)`**

Set the name of a stack.

Change the previous name of the stack S to a copy of NEW_NAME.

Note

Complexity: $O(1)$

Precondition

S must be a valid `gdsI_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

`gdsI_stack_get_name()` (p. 231)

4.17.3.11 void `gdsI_stack_set_growing_factor`(`gdsI_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 **`gdsI_stack_get_growing_factor()`** (p. 232)

Note

Complexity: $O(1)$

Precondition

S must be a valid `gdsI_stack_t`

Parameters

S	The stack to get the growing factor from
G	The new growing factor of S.

Returns

the growing factor of the stack S.

See also

`gdsI_stack_insert()` (p. 236)

`gdsI_stack_get_growing_factor()` (p. 232)

4.17.3.12 `gdsI_element_t` `gdsI_stack_insert`(`gdsI_stack_t` S, `void *` VALUE)

Insert an element in a stack (PUSH).

Allocate a new element E by calling S's `ALLOC_F` function on VALUE. `ALLOC_F` is the function pointer passed to **`gdsI_stack_alloc()`** (p. 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

S	The stack to insert in
VALUE	The value used to make the new element to insert into S

Returns

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

See also

`gdsI_stack_set_growing_factor()` (p. 236)
`gdsI_stack_get_growing_factor()` (p. 232)
`gdsI_stack_remove()` (p. 237)

Examples:

`examples/main_stack.c`.

4.17.3.13 `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

S	The stack to remove the top from
---	----------------------------------

Returns

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

See also

gdsI_stack_insert() (p. 236)

Examples:

examples/main_stack.c.

**4.17.3.14 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

S	The stack to search the element in
COMP_F	The comparison function used to compare S's element with VALUE
VALUE	The value to compare S's elements with

Returns

the first founded element E in case of success.
NULL if no element is found.

See also

gdsI_stack_search_by_position() (p. 238)

**4.17.3.15 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. 238)

Examples:

`examples/main_stack.c`.

4.17.3.16 `gdsl_element_t gdsl_stack_map_forward(const gdsl_stack_t S,
 gdsl_map_func_t MAP_F, void * USER_DATA)`

Parse a stack from bottom to top.

Parse all elements of the stack *S* from bottom to top. The *MAP_F* function is called on each *S*'s element with *USER_DATA* argument. If *MAP_F* returns *GDSL_MAP_S-STOP*, then **`gdsl_stack_map_forward()`** (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 <i>MAP_F</i> returns <i>GDSL_MAP_STOP</i> . Returns NULL when the parsing is done.

See also

gdsl_stack_map_backward() (p. 240)

Examples:

examples/main_stack.c.

4.17.3.17 `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. 240) stops and returns its last examined element.

Note

Complexity: $O(|S|)$

Precondition

S must be a valid gdsl_stack_t & MAP_F != NULL

Parameters

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

Returns

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

See also

gdsl_stack_map_forward() (p. 239)

4.17.3.18 `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. -
Additional USER_DATA argument could be passed to WRITE_F.

Note

Complexity: $O(|S|)$

Precondition

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

Parameters

<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. 241)

`gdsI_stack_dump()` (p. 242)

4.17.3.19 `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`. Additionnal `USER_DATA` argument could be passed to `WRITE_F`.

Note

Complexity: $O(|S|)$

Precondition

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

Parameters

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

See also

gdsl_stack_write() (p. 240)
gdsl_stack_dump() (p. 242)

Examples:

examples/main_stack.c.

4.17.3.20 void **gdsl_stack_dump**(**gdsl_stack_t** *S*, **gdsl_write_func_t** *WRITE_F*, FILE
 * *OUTPUT_FILE*, void * *USER_DATA*)

Dump the internal structure of a stack to a file.

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

Note

Complexity: $O(|S|)$

Precondition

S must be a valid **gdsl_stack_t** & *OUTPUT_FILE* != NULL

Parameters

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

See also

gdsl_stack_write() (p. 240)
gdsl_stack_write_xml() (p. 241)

Examples:

examples/main_stack.c.

4.18 GDSL types.

GDSL types.

Typedefs

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

Enumerations

- enum **gdsl_constant_t** { **GDSL_ERR_MEM_ALLOC** = -1, **GDSL_MAP_STOP** = 0, **GDSL_MAP_CONT** = 1, **GDSL_INSERTED**, **GDSL_FOUND** }
GDSL Constants.
- enum **gdsl_location_t** { **GDSL_LOCATION_UNDEF** = 0, **GDSL_LOCATION_HEAD** = 1, **GDSL_LOCATION_ROOT** = 1, **GDSL_LOCATION_TOP** = 1, **GDSL_LOCATION_TAIL** = 2, **GDSL_LOCATION_LEAF** = 2, **GDSL_LOCATION_BOTTOM** = 2, **GDSL_LOCATION_FIRST** = 1, **GDSL_LOCATION_LAST** = 2, **GDSL_LOCATION_FIRST_COL** = 1, **GDSL_LOCATION_LAST_COL** = 2, **GDSL_LOCATION_FIRST_ROW** = 4, **GDSL_LOCATION_LAST_ROW** = 8 }
- enum **bool** { **FALSE** = 0, **TRUE** = 1 }

4.18.1 Detailed Description

GDSL types.

4.18.2 Typedef Documentation

4.18.2.1 typedef void* gdsI_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 131 of file gdsI_types.h.

4.18.2.2 typedef gdsI_element_t(* gdsI_alloc_func_t)(void *USER_DATA)

GDSL Alloc element function type.

This function type is for allocating a new gdsI_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

gdsI_free_func_t (p. 244)

Definition at line 145 of file gdsI_types.h.

4.18.2.3 typedef void(* gdsI_free_func_t)(gdsI_element_t E)

GDSL Free element function type.

This function type is for freeing a gdsI_element_t variable. The element must have been previously allocated by a function of gdsI_alloc_func_t type. A free function according to gdsI_free_func_t must free the ressources allocated by the corresponding call to the function of type gdsI_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. 244)

Definition at line 163 of file `gdsl_types.h`.

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

GDSDL Copy element function type.

This function type is for copying `gdsl_element_t` variables.

Parameters

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

Returns

the copied element in case of success.

NULL in case of failure.

Definition at line 176 of file `gdsl_types.h`.

4.18.2.5 `typedef int(*gdsl_map_func_t)(const gdsl_element_t E, gdsl_location_t LOCATION, void *USER_DATA)`

GDSDL Map element function type.

This function type is for mapping a `gdsl_element_t` variable from a GDSDL 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

`GDSDL_MAP_STOP` if the mapping must be stopped.

`GDSDL_MAP_CONT` if the mapping must be continued.

Definition at line 193 of file `gdsl_types.h`.

4.18.2.6 `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 GDSL 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 greater than `VALUE`.

Definition at line 214 of file `gdsl_types.h`.

4.18.2.7 `typedef void(* gsdl_write_func_t)(const gsdl_element_t E, FILE *OUTPUT_FILE, gsdl_location_t LOCATION, void *USER_DATA)`

GDSL Write element function type.

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

Parameters

<i>E</i>	The <code>gdsl</code> 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 230 of file `gdsl_types.h`.

4.18.2.8 `typedef unsigned long int ulong`

Definition at line 243 of file `gdsl_types.h`.

4.18.2.9 `typedef unsigned short int ushort`

Definition at line 247 of file `gdsl_types.h`.

4.18.3 Enumeration Type Documentation

4.18.3.1 enum `gdsl_constant_t`

GDSDL Constants.

Enumerator:

GDSL_ERR_MEM_ALLOC Memory allocation error
GDSL_MAP_STOP For stopping a parsing function
GDSL_MAP_CONT For continuing a parsing function
GDSL_INSERTED To indicate an inserted value
GDSL_FOUND To indicate a founded value

Definition at line 49 of file `gdsl_types.h`.

4.18.3.2 enum `gdsl_location_t`

Enumerator:

GDSL_LOCATION_UNDEF Element position undefined
GDSL_LOCATION_HEAD Element is at head position
GDSL_LOCATION_ROOT Element is on leaf position
GDSL_LOCATION_TOP Element is at top position
GDSL_LOCATION_TAIL Element is at tail position
GDSL_LOCATION_LEAF Element is on root position
GDSL_LOCATION_BOTTOM Element is at bottom position
GDSL_LOCATION_FIRST Element is the first
GDSL_LOCATION_LAST Element is the last
GDSL_LOCATION_FIRST_COL Element is on first column
GDSL_LOCATION_LAST_COL Element is on last column
GDSL_LOCATION_FIRST_ROW Element is on first row
GDSL_LOCATION_LAST_ROW Element is on last row

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

4.18.3.3 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 268 of file `gdsl_types.h`.

Chapter 5

File Documentation

5.1 `_gdsl_bintree.h` File Reference

Low level binary tree manipulation module.

Typedefs

- `typedef struct _gdsl_bintree * _gdsl_bintree_t`
GDSL low-level binary tree type.
- `typedef int(* _gdsl_bintree_map_func_t)(const _gdsl_bintree_t TREE, void *USER_DATA)`
GDSL low-level binary tree map function type.
- `typedef void(* _gdsl_bintree_write_func_t)(const _gdsl_bintree_t TREE, FILE *OUTPUT_FILE, void *USER_DATA)`
GDSL low-level binary tree write function type.

Functions

- `_gdsl_bintree_t _gdsl_bintree_alloc` (const `gdsl_element_t` E, const `_gdsl_bintree_t` LEFT, const `_gdsl_bintree_t` RIGHT)
Create a new low-level binary tree.
- `void _gdsl_bintree_free` (`_gdsl_bintree_t` T, const `gdsl_free_func_t` FREE_F)
Destroy a low-level binary tree.
- `_gdsl_bintree_t _gdsl_bintree_copy` (const `_gdsl_bintree_t` T, const `gdsl_copy_func_t` COPY_F)
Copy a low-level binary tree.
- `bool _gdsl_bintree_is_empty` (const `_gdsl_bintree_t` T)
Check if a low-level binary tree is empty.
- `bool _gdsl_bintree_is_leaf` (const `_gdsl_bintree_t` T)

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

- **bool _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.

- **_gdsI_bintree_t _gdsI_bintree_rotate_right** (_gdsI_bintree_t *T)

Right rotate a low-level binary tree.

- **_gdsI_bintree_t _gdsI_bintree_rotate_left_right** (_gdsI_bintree_t *T)

Left-right rotate a low-level binary tree.

- **_gdsI_bintree_t _gdsI_bintree_rotate_right_left** (_gdsI_bintree_t *T)

Right-left rotate a low-level binary tree.

- **_gdsI_bintree_t _gdsI_bintree_map_prefix** (const _gdsI_bintree_t T, const _gdsI_bintree_map_func_t MAP_F, void *USER_DATA)

Parse a low-level binary tree in prefixed order.

- **_gdsI_bintree_t _gdsI_bintree_map_infix** (const _gdsI_bintree_t T, const _gdsI_bintree_map_func_t MAP_F, void *USER_DATA)

Parse a low-level binary tree in infix order.

- **_gdsI_bintree_t _gdsI_bintree_map_postfix** (const _gdsI_bintree_t T, const _gdsI_bintree_map_func_t MAP_F, void *USER_DATA)

Parse a low-level binary tree in postfix order.

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

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

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

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

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

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

5.1.1 Detailed Description

Low level binary tree manipulation module.

Definition in file `_gdsl_bintree.h`.

5.2 `_gdsl_bstree.h` File Reference

Low level binary search tree manipulation module.

Typedefs

- typedef **_gdsl_bintree_t _gdsl_bstree_t**
GDSL low-level binary search tree type.
- typedef int(* **_gdsl_bstree_map_func_t**)(**_gdsl_bstree_t** TREE, void *USER_DATA)
GDSL low-level binary search tree map function type.
- typedef void(* **_gdsl_bstree_write_func_t**)(**_gdsl_bstree_t** TREE, FILE *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 _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 **_gdsl_bstree_dump** (const **_gdsl_bstree_t** T, const **_gdsl_bstree_write_func_t** WRITE_F, FILE *OUTPUT_FILE, void *USER_DATA)

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

5.2.1 Detailed Description

Low level binary search tree manipulation module.

Definition in file **_gdsl_bstree.h**.

5.3 _gdsl_list.h File Reference

Low-level doubly-linked list manipulation module.

Typedefs

- typedef **_gdsl_node_t _gdsl_list_t**
GDSL low-level doubly-linked list type.

Functions

- **_gdsl_list_t _gdsl_list_alloc** (const **gdsl_element_t** E)
Create a new low-level list.
- void **_gdsl_list_free** (**_gdsl_list_t** L, const **gdsl_free_func_t** FREE_F)
Destroy a low-level list.
- **bool _gdsl_list_is_empty** (const **_gdsl_list_t** L)
Check if a low-level list is empty.
- **ulong _gdsl_list_get_size** (const **_gdsl_list_t** L)
Get the size of a low-level list.
- void **_gdsl_list_link** (**_gdsl_list_t** L1, **_gdsl_list_t** L2)
Link two low-level lists together.
- void **_gdsl_list_insert_after** (**_gdsl_list_t** L, **_gdsl_list_t** PREV)
Insert a low-level list after another one.
- void **_gdsl_list_insert_before** (**_gdsl_list_t** L, **_gdsl_list_t** SUCC)
Insert a low-level list before another one.
- void **_gdsl_list_remove** (**_gdsl_node_t** NODE)
Remove a node from a low-level list.
- **_gdsl_list_t _gdsl_list_search** (**_gdsl_list_t** L, const **gdsl_compare_func_t** COMP_F, void *VALUE)
Search for a particular node in a low-level list.
- **_gdsl_list_t _gdsl_list_map_forward** (const **_gdsl_list_t** L, const **_gdsl_node_map_func_t** MAP_F, void *USER_DATA)

Parse a low-level list in forward order.

- **_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.

5.3.1 Detailed Description

Low-level doubly-linked list manipulation module.

Definition in file **_gdsi_list.h**.

5.4 _gdsi_node.h File Reference

Low-level doubly-linked node manipulation module.

Typedefs

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

Functions

- **_gdsi_node_t _gdsi_node_alloc** (void)
Create a new low-level node.
- **_gdsi_element_t _gdsi_node_free** (**_gdsi_node_t** NODE)
Destroy a low-level node.
- **_gdsi_node_t _gdsi_node_get_succ** (const **_gdsi_node_t** NODE)
Get the successor of a low-level node.

- **gdsl_node_t gdsl_node_get_pred** (const **gdsl_node_t** NODE)
Get the predecessor of a low-level node.
- **gdsl_element_t gdsl_node_get_content** (const **gdsl_node_t** NODE)
Get the content of a low-level node.
- void **gdsl_node_set_succ** (**gdsl_node_t** NODE, const **gdsl_node_t** SUC-C)
Set the successor of a low-level node.
- void **gdsl_node_set_pred** (**gdsl_node_t** NODE, const **gdsl_node_t** PRE-D)
Set the predecessor of a low-level node.
- void **gdsl_node_set_content** (**gdsl_node_t** NODE, const **gdsl_element_t** -CONTENT)
Set the content of a low-level node.
- void **gdsl_node_link** (**gdsl_node_t** NODE1, **gdsl_node_t** NODE2)
Link two low-level nodes together.
- void **gdsl_node_unlink** (**gdsl_node_t** NODE1, **gdsl_node_t** NODE2)
Unlink two low-level nodes.
- void **gdsl_node_write** (const **gdsl_node_t** NODE, const **gdsl_node_write_func_t** WRITE_F, FILE *OUTPUT_FILE, void *USER_DATA)
Write a low-level node to a file.
- void **gdsl_node_write_xml** (const **gdsl_node_t** NODE, const **gdsl_node_write_func_t** WRITE_F, FILE *OUTPUT_FILE, void *USER_DATA)
Write a low-level node to a file into XML.
- void **gdsl_node_dump** (const **gdsl_node_t** NODE, const **gdsl_node_write_func_t** WRITE_F, FILE *OUTPUT_FILE, void *USER_DATA)
Dump the internal structure of a low-level node to a file.

5.4.1 Detailed Description

Low-level doubly-linked node manipulation module.

Definition in file **gdsl_node.h**.

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

2D-Arrays manipulation module.

Typedefs

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

Functions

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

5.6.1 Detailed Description

2D-Arrays manipulation module.

Definition in file **gdsi_2darray.h**.

5.7 gdsl_bstree.h File Reference

Binary search tree manipulation module.

Typedefs

- typedef struct gdsl_bstree * **gdsl_bstree_t**
GDSL binary search tree type.

Functions

- **gdsl_bstree_t gdsl_bstree_alloc** (const char *NAME, **gdsl_alloc_func_t** 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.

- **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.

5.7.1 Detailed Description

Binary search tree manipulation module.

Definition in file **gdsi_bstree.h**.

5.8 gdsi_hash.h File Reference

Hashtable manipulation module.

Typedefs

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

Functions

- **ulong gdsi_hash** (const char *KEY)
Computes a hash value from a NULL terminated character string.
- **gdsi_hash_t gdsi_hash_alloc** (const char *NAME, **gdsi_alloc_func_t** ALLOC_F, **gdsi_free_func_t** FREE_F, **gdsi_key_func_t** KEY_F, **gdsi_hash_func_t** HASH_F, **ushort** INITIAL_ENTRIES_NB)

Create a new hashtable.

- void **gdsl_hash_free** (gdsl_hash_t H)

Destroy a hashtable.

- void **gdsl_hash_flush** (gdsl_hash_t H)

Flush a hashtable.

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

Get the name of a hashtable.

- ushort **gdsl_hash_get_entries_number** (const gdsl_hash_t H)

Get the number of entries of a hashtable.

- ushort **gdsl_hash_get_lists_max_size** (const gdsl_hash_t H)

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

- ushort **gdsl_hash_get_longest_list_size** (const gdsl_hash_t H)

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

- ulong **gdsl_hash_get_size** (const gdsl_hash_t H)

Get the size of a hashtable.

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

Get the fill factor of a hashtable.

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

Set the name of a hashtable.

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

Insert an element into a hashtable (PUSH).

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

Remove an element from a hashtable (POP).

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

Delete an element from a hashtable.

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

Increase the dimensions of a hashtable.

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

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

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

Parse a hashtable.

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

Write all the elements of a hashtable to a file.

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

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

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

Dump the internal structure of a hashtable to a file.

5.8.1 Detailed Description

Hashtable manipulation module.

Definition in file **gdsl_hash.h**.

5.9 gdsl_heap.h File Reference

Heap manipulation module.

Typedefs

- typedef struct heap * **gdsl_heap_t**
GDsl heap type.

Functions

- **gdsl_heap_t gdsl_heap_alloc** (const char *NAME, **gdsl_alloc_func_t** ALLOC_F, **gdsl_free_func_t** FREE_F, **gdsl_compare_func_t** COMP_F)
Create a new heap.
- void **gdsl_heap_free** (**gdsl_heap_t** H)
Destroy a heap.
- void **gdsl_heap_flush** (**gdsl_heap_t** H)
Flush a heap.
- const char * **gdsl_heap_get_name** (const **gdsl_heap_t** H)
Get the name of a heap.
- **ulong gdsl_heap_get_size** (const **gdsl_heap_t** H)
Get the size of a heap.
- **gdsl_element_t gdsl_heap_get_top** (const **gdsl_heap_t** H)
Get the top of a heap.
- **bool gdsl_heap_is_empty** (const **gdsl_heap_t** H)
Check if a heap is empty.
- **gdsl_heap_t gdsl_heap_set_name** (**gdsl_heap_t** H, const char *NEW_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 **gdsl_heap_dump** (const **gdsl_heap_t** H, **gdsl_write_func_t** WRITE_F, FILE *OUTPUT_FILE, void *USER_DATA)
Dump the internal structure of a heap to a file.

5.9.1 Detailed Description

Heap manipulation module.

Definition in file **gdsl_heap.h**.

5.10 gdsl_interval_heap.h File Reference

Interval Heap manipulation module.

Typedefs

- typedef struct heap * **gdsl_interval_heap_t**
GDSL interval heap type.

Functions

- **gdsl_interval_heap_t gdsl_interval_heap_alloc** (const char *NAME, **gdsl_alloc_func_t** ALLOC_F, **gdsl_free_func_t** FREE_F, **gdsl_compare_func_t** 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 **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).
- **gdsI_element_t** **gdsI_interval_heap_get_min** (const **gdsI_interval_heap_t** H)
Get the minimum element.
- **gdsI_element_t** **gdsI_interval_heap_get_max** (const **gdsI_interval_heap_t** H)
Get the maximum element.
- **gdsI_interval_heap_t** **gdsI_interval_heap_delete_min** (**gdsI_interval_heap_t** H)
Delete the minimum element from an interval heap.
- **gdsI_interval_heap_t** **gdsI_interval_heap_delete_max** (**gdsI_interval_heap_t** H)
Delete the maximum element from an interval heap.
- **gdsI_element_t** **gdsI_interval_heap_map_forward** (const **gdsI_interval_heap_t** H, **gdsI_map_func_t** MAP_F, void *USER_DATA)
Parse a interval heap.
- void **gdsI_interval_heap_write** (const **gdsI_interval_heap_t** H, **gdsI_write_func_t** WRITE_F, FILE *OUTPUT_FILE, void *USER_DATA)
Write all the elements of an interval heap to a file.
- 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.
- 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.

5.10.1 Detailed Description

Interval Heap manipulation module.

Definition in file **gdsI_interval_heap.h**.

5.11 gdsl_list.h File Reference

Doubly-linked list manipulation module.

Typedefs

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

Functions

- **gdsl_list_t gdsl_list_alloc** (const char *NAME, **gdsl_alloc_func_t** ALLOC_F, **gdsl_free_func_t** FREE_F)
Create a new list.
- void **gdsl_list_free** (**gdsl_list_t** L)
Destroy a list.
- void **gdsl_list_flush** (**gdsl_list_t** L)
Flush a list.
- const char * **gdsl_list_get_name** (const **gdsl_list_t** L)
Get the name of a list.
- **ulong gdsl_list_get_size** (const **gdsl_list_t** L)
Get the size of a list.
- **bool gdsl_list_is_empty** (const **gdsl_list_t** L)
Check if a list is empty.
- **gdsl_element_t gdsl_list_get_head** (const **gdsl_list_t** L)
Get the head of a list.
- **gdsl_element_t gdsl_list_get_tail** (const **gdsl_list_t** L)
Get the tail of a list.
- **gdsl_list_t gdsl_list_set_name** (**gdsl_list_t** L, const char *NEW_NAME)
Set the name of a list.
- **gdsl_element_t gdsl_list_insert_head** (**gdsl_list_t** L, void *VALUE)
Insert an element at the head of a list.
- **gdsl_element_t gdsl_list_insert_tail** (**gdsl_list_t** L, void *VALUE)
Insert an element at the tail of a list.
- **gdsl_element_t gdsl_list_remove_head** (**gdsl_list_t** L)
Remove the head of a list.
- **gdsl_element_t gdsl_list_remove_tail** (**gdsl_list_t** L)
Remove the tail of a list.
- **gdsl_element_t gdsl_list_remove** (**gdsl_list_t** L, **gdsl_compare_func_t** COMP_F, const void *VALUE)
Remove a particular element from a list.

- **gdsl_list_t gsdl_list_delete_head** (gsdl_list_t L)
Delete the head of a list.
- **gdsl_list_t gsdl_list_delete_tail** (gsdl_list_t L)
Delete the tail of a list.
- **gsdl_list_t gsdl_list_delete** (gsdl_list_t L, gsdl_compare_func_t COMP_F, const void *VALUE)
Delete a particular element from a list.
- **gsdl_element_t gsdl_list_search** (const gsdl_list_t L, gsdl_compare_func_t COMP_F, const void *VALUE)
Search for a particular element into a list.
- **gsdl_element_t gsdl_list_search_by_position** (const gsdl_list_t L, ulong POS)
Search for an element by its position in a list.
- **gsdl_element_t gsdl_list_search_max** (const gsdl_list_t L, gsdl_compare_func_t COMP_F)
Search for the greatest element of a list.
- **gsdl_element_t gsdl_list_search_min** (const gsdl_list_t L, gsdl_compare_func_t COMP_F)
Search for the lowest element of a list.
- **gsdl_list_t gsdl_list_sort** (gsdl_list_t L, gsdl_compare_func_t COMP_F)
Sort a list.
- **gsdl_element_t gsdl_list_map_forward** (const gsdl_list_t L, gsdl_map_func_t MAP_F, void *USER_DATA)
Parse a list from head to tail.
- **gsdl_element_t gsdl_list_map_backward** (const gsdl_list_t L, gsdl_map_func_t MAP_F, void *USER_DATA)
Parse a list from tail to head.
- void **gsdl_list_write** (const gsdl_list_t L, gsdl_write_func_t WRITE_F, FILE *OUTPUT_FILE, void *USER_DATA)
Write all the elements of a list to a file.
- void **gsdl_list_write_xml** (const gsdl_list_t L, gsdl_write_func_t WRITE_F, FILE *OUTPUT_FILE, void *USER_DATA)
Write the content of a list to a file into XML.
- void **gsdl_list_dump** (const gsdl_list_t L, gsdl_write_func_t WRITE_F, FILE *OUTPUT_FILE, void *USER_DATA)
Dump the internal structure of a list to a file.
- **gsdl_list_cursor_t gsdl_list_cursor_alloc** (const gsdl_list_t L)
Create a new list cursor.
- void **gsdl_list_cursor_free** (gsdl_list_cursor_t C)
Destroy a list cursor.
- void **gsdl_list_cursor_move_to_head** (gsdl_list_cursor_t C)
Put a cursor on the head of its list.
- void **gsdl_list_cursor_move_to_tail** (gsdl_list_cursor_t C)
Put a cursor on the tail of its list.

- **gdsl_element_t gdsl_list_cursor_move_to_value** (gdsl_list_cursor_t C, gdsl_compare_func_t COMP_F, void *VALUE)
Place a cursor on a particular element.
- **gdsl_element_t gdsl_list_cursor_move_to_position** (gdsl_list_cursor_t C, ulong POS)
Place a cursor on a element given by its position.
- **void gdsl_list_cursor_step_forward** (gdsl_list_cursor_t C)
Move a cursor one step forward of its list.
- **void gdsl_list_cursor_step_backward** (gdsl_list_cursor_t C)
Move a cursor one step backward of its list.
- **bool gdsl_list_cursor_is_on_head** (const gdsl_list_cursor_t C)
Check if a cursor is on the head of its list.
- **bool gdsl_list_cursor_is_on_tail** (const gdsl_list_cursor_t C)
Check if a cursor is on the tail of its list.
- **bool gdsl_list_cursor_has_succ** (const gdsl_list_cursor_t C)
Check if a cursor has a successor.
- **bool gdsl_list_cursor_has_pred** (const gdsl_list_cursor_t C)
Check if a cursor has a predecessor.
- **void gdsl_list_cursor_set_content** (gdsl_list_cursor_t C, gdsl_element_t E)
Set the content of the cursor.
- **gdsl_element_t gdsl_list_cursor_get_content** (const gdsl_list_cursor_t C)
Get the content of a cursor.
- **gdsl_element_t gdsl_list_cursor_insert_after** (gdsl_list_cursor_t C, void *VALUE)
Insert a new element after a cursor.
- **gdsl_element_t gdsl_list_cursor_insert_before** (gdsl_list_cursor_t C, void *VALUE)
Insert a new element before a cursor.
- **gdsl_element_t gdsl_list_cursor_remove** (gdsl_list_cursor_t C)
Remove the element under a cursor.
- **gdsl_element_t gdsl_list_cursor_remove_after** (gdsl_list_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.

5.11.1 Detailed Description

Doubly-linked list manipulation module.

Definition in file **gdsl_list.h**.

5.12 **gdsl_macros.h** File Reference

Various macros module.

Defines

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

5.12.1 Detailed Description

Various macros module.

Definition in file **gdsl_macros.h**.

5.13 **gdsl_perm.h** File Reference

Permutation manipulation module.

Typedefs

- **typedef struct gdsl_perm * gdsl_perm_t**
GDSL permutation type.
- **typedef void(* gdsl_perm_write_func_t)(ulong E, FILE *OUTPUT_FILE, gdsl_location_t POSITION, void *USER_DATA)**
GDSL permutation write function type.
- **typedef struct gdsl_perm_data * gdsl_perm_data_t**

Enumerations

- **enum gdsl_perm_position_t { GDSL_PERM_POSITION_FIRST = 1, GDSL_PERM_POSITION_LAST = 2 }**
This type is for gdsl_perm_write_func_t.

Functions

- **gdsl_perm_t gdsl_perm_alloc** (const char *NAME, const **ulong** N)
Create a new permutation.
- **void gdsl_perm_free** (gdsl_perm_t P)
Destroy a permutation.
- **gdsl_perm_t gdsl_perm_copy** (const gdsl_perm_t P)
Copy a permutation.
- **const char *****gdsl_perm_get_name** (const gdsl_perm_t P)
Get the name of a permutation.
- **ulong gdsl_perm_get_size** (const gdsl_perm_t P)
Get the size of a permutation.
- **ulong gdsl_perm_get_element** (const gdsl_perm_t P, const **ulong** INDIX)
Get the (INDIX+1)-th element from a permutation.
- **ulong *****gdsl_perm_get_elements_array** (const gdsl_perm_t P)
Get the array elements of a permutation.
- **ulong gdsl_perm_linear_inversions_count** (const gdsl_perm_t P)
Count the inversions number into a linear permutation.
- **ulong gdsl_perm_linear_cycles_count** (const gdsl_perm_t P)
Count the cycles number into a linear permutation.
- **ulong gdsl_perm_canonical_cycles_count** (const gdsl_perm_t P)
Count the cycles number into a canonical permutation.
- **gdsl_perm_t gdsl_perm_set_name** (gdsl_perm_t P, const char *NEW_NAME)
Set the name of a permutation.
- **gdsl_perm_t gdsl_perm_linear_next** (gdsl_perm_t P)
Get the next permutation from a linear permutation.
- **gdsl_perm_t gdsl_perm_linear_prev** (gdsl_perm_t P)
Get the previous permutation from a linear permutation.
- **gdsl_perm_t gdsl_perm_set_elements_array** (gdsl_perm_t P, const **ulong** *ARRAY)
Initialize a permutation with an array of values.
- **gdsl_perm_t gdsl_perm_multiply** (gdsl_perm_t RESULT, const gdsl_perm_t ALPHA, const gdsl_perm_t BETA)
Multiply two permutations.
- **gdsl_perm_t gdsl_perm_linear_to_canonical** (gdsl_perm_t Q, const gdsl_perm_t P)
Convert a linear permutation to its canonical form.
- **gdsl_perm_t gdsl_perm_canonical_to_linear** (gdsl_perm_t Q, const gdsl_perm_t P)
Convert a canonical permutation to its linear form.
- **gdsl_perm_t gdsl_perm_inverse** (gdsl_perm_t P)
Inverse in place a permutation.
- **gdsl_perm_t gdsl_perm_reverse** (gdsl_perm_t P)

Reverse in place a permutation.

- **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.

5.13.1 Detailed Description

Permutation manipulation module.

Definition in file **gdsI_perm.h**.

5.14 gdsI_queue.h File Reference

Queue manipulation module.

Typedefs

- typedef struct _gdsI_queue * **gdsI_queue_t**
GDSL queue type.

Functions

- **gdsI_queue_t gdsI_queue_alloc** (const char *NAME, gdsI_alloc_func_t ALL_OC_F, gdsI_free_func_t FREE_F)
Create a new queue.
- void **gdsI_queue_free** (gdsI_queue_t Q)
Destroy a queue.
- void **gdsI_queue_flush** (gdsI_queue_t Q)
Flush a queue.
- const char * **gdsI_queue_get_name** (const gdsI_queue_t Q)
Get the name of a queue.
- **ulong gdsI_queue_get_size** (const gdsI_queue_t Q)

Get the size of a queue.

- **bool gdsl_queue_is_empty** (const **gdsl_queue_t** Q)

Check if a queue is empty.

- **gdsl_element_t gdsl_queue_get_head** (const **gdsl_queue_t** Q)

Get the head of a queue.

- **gdsl_element_t gdsl_queue_get_tail** (const **gdsl_queue_t** Q)

Get the tail of a queue.

- **gdsl_queue_t gdsl_queue_set_name** (**gdsl_queue_t** Q, const char *NEW_NAME)

Set the name of a queue.

- **gdsl_element_t gdsl_queue_insert** (**gdsl_queue_t** Q, void *VALUE)

Insert an element in a queue (PUT).

- **gdsl_element_t gdsl_queue_remove** (**gdsl_queue_t** Q)

Remove an element from a queue (GET).

- **gdsl_element_t gdsl_queue_search** (const **gdsl_queue_t** Q, **gdsl_compare_func_t** COMP_F, void *VALUE)

Search for a particular element in a queue.

- **gdsl_element_t gdsl_queue_search_by_position** (const **gdsl_queue_t** Q, **ulong** POS)

Search for an element by its position in a queue.

- **gdsl_element_t gdsl_queue_map_forward** (const **gdsl_queue_t** Q, **gdsl_map_func_t** MAP_F, void *USER_DATA)

Parse a queue from head to tail.

- **gdsl_element_t gdsl_queue_map_backward** (const **gdsl_queue_t** Q, **gdsl_map_func_t** MAP_F, void *USER_DATA)

Parse a queue from tail to head.

- void **gdsl_queue_write** (const **gdsl_queue_t** Q, **gdsl_write_func_t** WRITE_F, FILE *OUTPUT_FILE, void *USER_DATA)

Write all the elements of a queue to a file.

- void **gdsl_queue_write_xml** (const **gdsl_queue_t** Q, **gdsl_write_func_t** WRITE_F, FILE *OUTPUT_FILE, void *USER_DATA)

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

- void **gdsl_queue_dump** (const **gdsl_queue_t** Q, **gdsl_write_func_t** WRITE_F, FILE *OUTPUT_FILE, void *USER_DATA)

Dump the internal structure of a queue to a file.

5.14.1 Detailed Description

Queue manipulation module.

Definition in file **gdsl_queue.h**.

5.15 gdsl_rbtrees.h File Reference

Red-black tree manipulation module.

Typedefs

- typedef struct gdsl_rbtrees * **gdsl_rbtrees_t**

Functions

- **gdsl_rbtrees_t gdsl_rbtrees_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 red-black tree.
- void **gdsl_rbtrees_free** (**gdsl_rbtrees_t** T)
Destroy a red-black tree.
- void **gdsl_rbtrees_flush** (**gdsl_rbtrees_t** T)
Flush a red-black tree.
- char * **gdsl_rbtrees_get_name** (const **gdsl_rbtrees_t** T)
Get the name of a red-black tree.
- bool **gdsl_rbtrees_is_empty** (const **gdsl_rbtrees_t** T)
Check if a red-black tree is empty.
- **gdsl_element_t gdsl_rbtrees_get_root** (const **gdsl_rbtrees_t** T)
Get the root of a red-black tree.
- **ulong gdsl_rbtrees_get_size** (const **gdsl_rbtrees_t** T)
Get the size of a red-black tree.
- **ulong gdsl_rbtrees_height** (const **gdsl_rbtrees_t** T)
Get the height of a red-black tree.
- **gdsl_rbtrees_t gdsl_rbtrees_set_name** (**gdsl_rbtrees_t** T, const char *NEW_NAME)
Set the name of a red-black tree.
- **gdsl_element_t gdsl_rbtrees_insert** (**gdsl_rbtrees_t** T, void *VALUE, int *RESULT)
Insert an element into a red-black tree if it's not found or return it.
- **gdsl_element_t gdsl_rbtrees_remove** (**gdsl_rbtrees_t** T, void *VALUE)
Remove an element from a red-black tree.
- **gdsl_rbtrees_t gdsl_rbtrees_delete** (**gdsl_rbtrees_t** T, void *VALUE)
Delete an element from a red-black tree.
- **gdsl_element_t gdsl_rbtrees_search** (const **gdsl_rbtrees_t** T, **gdsl_compare_func_t** COMP_F, void *VALUE)
Search for a particular element into a red-black tree.
- **gdsl_element_t gdsl_rbtrees_map_prefix** (const **gdsl_rbtrees_t** T, **gdsl_map_func_t** MAP_F, void *USER_DATA)
Parse a red-black tree in prefixed order.
- **gdsl_element_t gdsl_rbtrees_map_infix** (const **gdsl_rbtrees_t** T, **gdsl_map_func_t** MAP_F, void *USER_DATA)
Parse a red-black tree in infix order.
- **gdsl_element_t gdsl_rbtrees_map_postfix** (const **gdsl_rbtrees_t** T, **gdsl_map_func_t** MAP_F, void *USER_DATA)

Parse a red-black tree in postfix order.

- void **gdsl_rbtrees_write** (const **gdsl_rbtrees_t** T, **gdsl_write_func_t** WRITE_F, FILE *OUTPUT_FILE, void *USER_DATA)

Write the element of each node of a red-black tree to a file.

- void **gdsl_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.

5.15.1 Detailed Description

Red-black tree manipulation module.

Definition in file **gdsl_rbtrees.h**.

5.16 gdsl_sort.h File Reference

Sort module.

Functions

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

Sort an array in place.

5.16.1 Detailed Description

Sort module.

Definition in file **gdsl_sort.h**.

5.17 gdsl_stack.h File Reference

Stack manipulation module.

Typedefs

- typedef struct **_gdsl_stack** * **gdsl_stack_t**

GDSL stack type.

Functions

- **gdsl_stack_t gdsl_stack_alloc** (const char *NAME, **gdsl_alloc_func_t** ALLOC_F, **gdsl_free_func_t** FREE_F)
Create a new stack.
- **void gdsl_stack_free** (**gdsl_stack_t** S)
Destroy a stack.
- **void gdsl_stack_flush** (**gdsl_stack_t** S)
Flush a stack.
- **const char * gdsl_stack_get_name** (const **gdsl_stack_t** S)
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 **gdsI_stack_write_xml**(**gdsI_stack_t** S, **gdsI_write_func_t** WRITE_F, FILE *OUTPUT_FILE, void *USER_DATA)

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

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

Dump the internal structure of a stack to a file.

5.17.1 Detailed Description

Stack manipulation module.

Definition in file **gdsI_stack.h**.

5.18 gdsI_types.h File Reference

GDSL types.

Typedefs

- typedef void * **gdsI_element_t**
GDSL element type.
- typedef **gdsI_element_t**(* **gdsI_alloc_func_t**)(void *USER_DATA)
GDSL Alloc element function type.
- typedef void(* **gdsI_free_func_t**)(**gdsI_element_t** E)
GDSL Free element function type.
- typedef **gdsI_element_t**(* **gdsI_copy_func_t**)(const **gdsI_element_t** E)
GDSL Copy element function type.
- typedef int(* **gdsI_map_func_t**)(const **gdsI_element_t** E, **gdsI_location_t** LOCATION, void *USER_DATA)
GDSL Map element function type.
- typedef long int(* **gdsI_compare_func_t**)(const **gdsI_element_t** E, void *VALUE)
GDSL Comparison element function type.
- typedef void(* **gdsI_write_func_t**)(const **gdsI_element_t** E, FILE *OUTPUT_FILE, **gdsI_location_t** LOCATION, void *USER_DATA)
GDSL Write element function type.
- typedef unsigned long int **ulong**
- typedef unsigned short int **ushort**

Enumerations

- enum **gdsi_constant_t** { **GDSL_ERR_MEM_ALLOC** = -1, **GDSL_MAP_STOP** = 0, **GDSL_MAP_CONT** = 1, **GDSL_INSERTED**, **GDSL_FOUND** }

GDSL Constants.

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

5.18.1 Detailed Description

GDSL types.

Definition in file **gdsi_types.h**.

5.19 mainpage.h File Reference

Chapter 6

Example Documentation

6.1 examples/main_2darray.c

This is an example of how to use `gdsl_2darray` module.

```
/*
 * This file is part of the Generic Data Structures Library (GDSL).
 * Copyright (C) 1998-2018 Nicolas Darnis <ndarnis@free.fr>.
 *
 * GDSL is free software: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * GDSL is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with GDSL. If not, see <http://www.gnu.org/licenses/>.
 *
 *
 * GDSL - Generic Data Structures Library
 * $RCSfile: main_2darray.c,v $
 * $Revision: 1.12 $
 * $Date: 2015/02/17 12:33:16 $
 */

#include <config.h>

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

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

```
#define ROWS_NB 4UL
#define COLS_NB 3UL

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

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

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

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

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

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

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

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

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

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

    gdsl_2darray_free (m);
}
```

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

6.2 examples/main_bstree.c

This is an example of how to use `gdsl_bstree` module.

```
/*  
 * This file is part of the Generic Data Structures Library (GDSL).  
 * Copyright (C) 1998-2018 Nicolas Darnis <ndarnis@free.fr>.  
 *  
 * GDSL is free software: you can redistribute it and/or modify  
 * it under the terms of the GNU General Public License as published by  
 * the Free Software Foundation, either version 3 of the License, or  
 * (at your option) any later version.  
 *  
 * GDSL is distributed in the hope that it will be useful,  
 * but WITHOUT ANY WARRANTY; without even the implied warranty of  
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the  
 * GNU General Public License for more details.  
 *  
 * You should have received a copy of the GNU General Public License  
 * along with GDSL. If not, see <http://www.gnu.org/licenses/>.  
 *  
 *  
 * GDSL - Generic Data Structures Library  
 * $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];  
    gdsl_bstree_t t = gdsl_bstree_alloc ("MY BSTREE", alloc_string, free_string  
        , compare_strings);
```

```

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

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

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

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

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

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

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

    case 3:

```

```
        gdsl_bstree_flush (t);
        break;

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

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

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

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

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

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

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

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

    case 11:
        {
            int i;
            int rc;
```

```

        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);
}

```

6.3 examples/main_hash.c

This is an example of how to use gds1_hash module.

```

/*
 * This file is part of the Generic Data Structures Library (GDSL).
 * Copyright (C) 1998-2018 Nicolas Darnis <ndarnis@free.fr>.
 *
 * GDSL is free software: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * GDSL is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with GDSL. If not, see <http://www.gnu.org/licenses/>.
 *
 *
 * GDSL - Generic Data Structures Library
 * $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);
}
```

6.4 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 (GDSL).
 * Copyright (C) 1998-2018 Nicolas Darnis <ndarnis@free.fr>.
 *
 * GDSL is free software: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * GDSL is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with GDSL. If not, see <http://www.gnu.org/licenses/>.
 *
 *
 * GDSL - Generic Data Structures Library
 * $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%d ",
            (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);
            gds1_heap_insert (h, (void*) &value);
        }
        break;

    case 2:
        if (!gds1_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*) gdsl_heap_set_top (h, (void*) &value);
if (v == NULL)
{
    printf ("value is greather than all other heap ones\n");
}
else
{
    printf ("old value was: %ld\n", *v);
    free_integer (v);
}
}
break;

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

    value = (long int*) gdsl_heap_remove (h, &pos);
    if (value == NULL)
    {
        printf ("Not found\n");
    }
    else
    {
        printf ("Value removed %ld\n", *value);
        free_integer (value);
    }
}
break;
*/
case 7:
if (gdsl_heap_is_empty (h))
{
    printf ("The heap '%s' is empty\n", gdsl_heap_get_name (h));
}
else
{
    printf ("%s = ( ", gdsl_heap_get_name (h));
    gdsl_heap_map_forward (h, my_display_integer, NULL);
    printf (")\n");
}
break;

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

case 9:
```

```

        gds1_heap_write_xml (h, print_integer, stdout, NULL);
        break;
    }
}
while (choix != 0);

gds1_heap_free (h);

exit (EXIT_SUCCESS);
}

```

6.5 examples/main_interval_heap.c

This is an example of how to use gds1_interval_heap module.

```

/*
 * This file is part of the Generic Data Structures Library (GDSL).
 * Copyright (C) 1998-2018 Nicolas Darnis <ndarnis@free.fr>.
 *
 * GDSL is free software: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * GDSL is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with GDSL. If not, see <http://www.gnu.org/licenses/>.
 *
 * GDSL - Generic Data Structures Library
 * $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 "gds1_types.h"
#include "gds1_interval_heap.h"

#include "_integers.h"

static int
my_display_integer (const gds1_element_t e, gds1_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);
    gsdl_interval_heap_insert(h, (void *) value);
    //gsdl_raw_heap_dump(h);
    //gsdl_check_interval_heap_integrity(h);
}

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

    return value;
}

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

    return value;
}

void test1() {
    gsdl_interval_heap_t h = gsdl_interval_heap_alloc ("H", alloc_integer,
        free_integer, compare_integers);

    insert_value(h, 2);
    insert_value(h, 30);
    insert_value(h, 3);
    insert_value(h, 20);
    insert_value(h, 4);
    insert_value(h, 25);
    insert_value(h, 8);
    insert_value(h, 16);
    insert_value(h, 4);
    //exit(0);
    insert_value(h, 10);
    insert_value(h, 10);
    insert_value(h, 15);
}

```

```

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

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

    remove_min(h);

    remove_max(h);

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

    gdsl_interval_heap_flush(h);

    assert(gdsl_interval_heap_get_size(h) == 0);

    gdsl_interval_heap_free (h);
}

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

    gdsl_interval_heap_flush(h);

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

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

    assert(gdsl_interval_heap_get_size(h) == 0);

    gdsl_interval_heap_free (h);
}

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

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

void test3() {

```

```

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

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

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

    removed[i] = e;
}

check_removed(removed, len/2);

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

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

    removed[i] = e;
}

check_removed(removed, len/2);

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

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

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

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

```

```

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

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

```

6.6 examples/main_list.c

This is an example of how to use `gdsl_list` module.

```

/*
 * This file is part of the Generic Data Structures Library (GDSL).
 * Copyright (C) 1998-2018 Nicolas Darnis <ndarnis@free.fr>.
 *
 * GDSL is free software: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * GDSL is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with GDSL. If not, see <http://www.gnu.org/licenses/>.
 *
 *
 * GDSL - Generic Data Structures Library
 * $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 (<-) = ( ", gdsl_list_get_name (l));

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

    printf (")\n");
}

int main (void)
{
    int choix = 0;

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

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

```

```

printf ("\t 0> Quit\n\n");
printf ("\t\tYour choice: ");
scanf ("%d", &choix);

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

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

do
{
int choix;

printf ("\t\tMENU - CELL INSERTION\n\n");
printf ("\t1> Insert cell at the beginning of the list\n");
printf ("\t2> Insert cell at end of list\n");
printf ("\t3> Insert cell after another cell\n");
printf ("\t4> Insert cell before another cell\n");
printf ("\t5> Display the list\n");
printf ("\t0> RETURN TO MAIN MENU\n\n");
printf ("\t\tYour choice: ");
scanf ("%d", &choix );

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

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

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

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

scanf ("%s", Nom);

if (!gdsl_list_cursor_move_to_value (c, compare_strings
, Nom))
{
printf ("The cell '%s' doesn't exist\n", Nom);
}
}
}
}
}
}

```

```

        }
        else
        {
            gdsl_list_cursor_insert_after (c, nom);
            done = 1;
        }
        gdsl_list_cursor_free (c);
    }
    break;

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

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

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

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

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

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

```

```
    }
    else
    {
        gdsl_list_delete_head (l);
    }
    break;

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

case 4:
    {
        char nom[100];

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

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

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

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

```
    }
    break;

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

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

case 9:
    if (gdsl_list_is_empty (l))
    {
        printf ("The list is empty.\n");
    }
    else
    {
        gdsl_list_dump (l, print_string, stdout, NULL);
    }
    break;

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

case 11:
    {
        int pos;
        gdsl_element_t e;

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

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

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

```

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

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

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

case 14:
    if (gdsl_list_is_empty (l))
    {
        printf ("The list is empty.\n");
    }
    else
    {
        printf ("Max Element: %s\n", (char*) gdsl_list_search_max (l,
compare_strings));
    }
    break;

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

    gdsl_perm_randomize (p);

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

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

{
    int i = 0;

```

```

        gdslist_cursor_t c = gdslist_cursor_alloc (1);

        for (gdslist_cursor_move_to_head (c); gdslist_cursor_get_content
(c); gdslist_cursor_step_forward (c))
        {
            char toto[50];
            sprintf (toto, "%d", i++);

            gdslist_cursor_insert_before (c, toto);

            gdslist_cursor_step_backward (c);
            gdslist_cursor_delete_after (c);
        }

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

gdslist_free (l);

exit (EXIT_SUCCESS);
}

```

6.7 examples/main_llbintree.c

This is an example of how to use `_gdslist_bintree` module.

```

/*
 * This file is part of the Generic Data Structures Library (GDSL).
 * Copyright (C) 1998-2018 Nicolas Darnis <ndarnis@free.fr>.
 *
 * GDSL is free software: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * GDSL is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with GDSL. If not, see <http://www.gnu.org/licenses/>.
 *
 *
 * GDSL - Generic Data Structures Library
 * $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);
}

```

6.8 examples/main_llbtree.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) 1998-2018 Nicolas Darnis <ndarnis@free.fr>.
 *
 * GDSL is free software: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * GDSL is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with GDSL. If not, see <http://www.gnu.org/licenses/>.
 *
 * GDSL - Generic Data Structures Library
 * $RCSfile: main_llbtree.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);
}

```

6.9 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 (GDSDL).
 * Copyright (C) 1998-2018 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/>.
 *
 *
 * GDSDL - Generic Data Structures Library
 * $RCSfile: main_lllist.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 GDSDL_MAP_CONT;
}

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

```

```

_gdsl_list_t b = _gdsl_list_alloc (alloc_string ("b"));
_gdsl_list_t c = _gdsl_list_alloc (alloc_string ("c"));

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

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

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

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

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

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

_gdsl_list_free (a, free_string);

exit (EXIT_SUCCESS);
}

```

6.10 examples/main_perm.c

This is an example of how to use `gdsl_perm` module.

```

/*
 * This file is part of the Generic Data Structures Library (GDSL).
 * Copyright (C) 1998-2018 Nicolas Darnis <ndarnis@free.fr>.
 *
 * GDSL is free software: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * GDSL is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with GDSL. If not, see <http://www.gnu.org/licenses/>.
 *
 *
 * GDSL - Generic Data Structures Library
 * $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 "gdsl_types.h"
#include "gdsl_perm.h"
#include "_integers.h"

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

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

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

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

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

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

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

    n = atoi (argv[1]);

    c_alpha = gdsl_perm_alloc ("c_alpha", n);

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

```

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

gdsl_perm_reverse (l_alpha);

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

gdsl_perm_reverse (l_alpha);
gdsl_perm_inverse (l_alpha);

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

gdsl_perm_inverse (l_alpha);

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

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

gdsl_perm_canonical_to_linear (l_alpha, c_alpha);

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

gdsl_perm_free (l_alpha);
gdsl_perm_free (c_alpha);

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

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

  gdsl_perm_randomize (a);

```

```

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

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

exit (EXIT_SUCCESS);
}

```

6.11 examples/main_queue.c

This is an example of how to use `gdsl_queue` module.

```

/*
 * This file is part of the Generic Data Structures Library (GDSL).
 * Copyright (C) 1998-2018 Nicolas Darnis <ndarnis@free.fr>.
 *
 * GDSL is free software: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * GDSL is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with GDSL. If not, see <http://www.gnu.org/licenses/>.
 *
 *
 * GDSL - Generic Data Structures Library
 * $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", gdsl_queue_get_name (q));
        }
    }
    break;

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

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

        value = (int*) gdsl_queue_search_by_position (q, pos);
```

```

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

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

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

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

gdsl_queue_free (q);

exit (EXIT_SUCCESS);
}

```

6.12 examples/main_rbtrees.c

This is an example of how to use gdsl_rbtrees module.

```

/*
 * This file is part of the Generic Data Structures Library (GDSDL).
 * Copyright (C) 1998-2018 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/>.
 *
 */

```

```

* GDSL - Generic Data Structures Library
* $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 "gdsl_perm.h"
#include "gdsl_types.h"
#include "gdsl_rbtrees.h"
#include "_strings.h"
#include "_integers.h"

#define N 100

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

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

    do
    {
        printf ("\t\tMENU - RBTTREE\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_rbtrees_insert (t, (void *) name, &rc);

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

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

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

        if (gdsl_rbtrees_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,
GDSDL_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);
}

```

6.13 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) 1998-2018 Nicolas Darnis <ndarnis@free.fr>.
 *
 * GDSL is free software: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * GDSL is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with GDSL. If not, see <http://www.gnu.org/licenses/>.
 *
 *
 * GDSL - Generic Data Structures Library
 * $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);
}

```

6.14 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 (GDSDL).
 * Copyright (C) 1998-2018 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/>.
*
*
* GDSDL - Generic Data Structures Library
* $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 GDSDL_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.,

84

- gdsl_bstree_alloc, 85
- gdsl_bstree_delete, 92
- gdsl_bstree_dump, 98
- gdsl_bstree_flush, 87
- gdsl_bstree_free, 86
- gdsl_bstree_get_height, 90
- gdsl_bstree_get_name, 87
- gdsl_bstree_get_root, 89
- gdsl_bstree_get_size, 89
- gdsl_bstree_insert, 91
- gdsl_bstree_is_empty, 88
- gdsl_bstree_map_infix, 95
- gdsl_bstree_map_postfix, 95
- gdsl_bstree_map_prefix, 94
- gdsl_bstree_remove, 92
- gdsl_bstree_search, 93
- gdsl_bstree_set_name, 90
- gdsl_bstree_t, 85
- gdsl_bstree_write, 96
- gdsl_bstree_write_xml, 97

Doubly-linked list manipulation module.,

142

- gdsl_list_alloc, 145
- gdsl_list_cursor_alloc, 163
- gdsl_list_cursor_delete, 175
- gdsl_list_cursor_delete_after, 175
- gdsl_list_cursor_delete_before, 176
- gdsl_list_cursor_free, 164
- gdsl_list_cursor_get_content, 171
- gdsl_list_cursor_has_pred, 170
- gdsl_list_cursor_has_succ, 169
- gdsl_list_cursor_insert_after, 171
- gdsl_list_cursor_insert_before, 172
- gdsl_list_cursor_is_on_head, 168
- gdsl_list_cursor_is_on_tail, 168
- gdsl_list_cursor_move_to_head, 164
- gdsl_list_cursor_move_to_position, 166
- gdsl_list_cursor_move_to_tail, 165

- gdsl_list_cursor_move_to_value, 165

- gdsl_list_cursor_remove, 173
- gdsl_list_cursor_remove_after, 173
- gdsl_list_cursor_remove_before, 174
- gdsl_list_cursor_set_content, 170
- gdsl_list_cursor_step_backward, 167
- gdsl_list_cursor_step_forward, 167
- gdsl_list_cursor_t, 145
- gdsl_list_delete, 155
- gdsl_list_delete_head, 154
- gdsl_list_delete_tail, 155
- gdsl_list_dump, 163
- gdsl_list_flush, 146
- gdsl_list_free, 146
- gdsl_list_get_head, 149
- gdsl_list_get_name, 147
- gdsl_list_get_size, 148
- gdsl_list_get_tail, 149
- gdsl_list_insert_head, 150
- gdsl_list_insert_tail, 151
- gdsl_list_is_empty, 148
- gdsl_list_map_backward, 160
- gdsl_list_map_forward, 160
- gdsl_list_remove, 153
- gdsl_list_remove_head, 152
- gdsl_list_remove_tail, 153
- gdsl_list_search, 156
- gdsl_list_search_by_position, 157
- gdsl_list_search_max, 158
- gdsl_list_search_min, 158
- gdsl_list_set_name, 150
- gdsl_list_sort, 159
- gdsl_list_t, 145
- gdsl_list_write, 161
- gdsl_list_write_xml, 162

FALSE

- GDSL types., 247

GDSL types., 243

- FALSE, 247

- GDSL_ERR_MEM_ALLOC, 247

- GDSL_FOUND, 247
- GDSL_INSERTED, 247
- GDSL_LOCATION_BOTTOM, 247
- GDSL_LOCATION_FIRST, 247
- GDSL_LOCATION_FIRST_COL, 247
- GDSL_LOCATION_FIRST_ROW, 247
- GDSL_LOCATION_HEAD, 247
- GDSL_LOCATION_LAST, 247
- GDSL_LOCATION_LAST_COL, 247
- GDSL_LOCATION_LAST_ROW, 247
- GDSL_LOCATION_LEAF, 247
- GDSL_LOCATION_ROOT, 247
- GDSL_LOCATION_TAIL, 247
- GDSL_LOCATION_TOP, 247
- GDSL_LOCATION_UNDEF, 247
- GDSL_MAP_CONT, 247
- GDSL_MAP_STOP, 247
- TRUE, 247
- bool, 247
- gds_alloc_func_t, 244
- gds_compare_func_t, 245
- gds_constant_t, 246
- gds_copy_func_t, 245
- gds_element_t, 244
- gds_free_func_t, 244
- gds_location_t, 247
- gds_map_func_t, 245
- gds_write_func_t, 246
- ulong, 246
- ushort, 246
- GDSL_ERR_MEM_ALLOC
 - GDSL types., 247
- GDSL_FOUND
 - GDSL types., 247
- GDSL_INSERTED
 - GDSL types., 247
- GDSL_LOCATION_BOTTOM
 - GDSL types., 247
- GDSL_LOCATION_FIRST
 - GDSL types., 247
- GDSL_LOCATION_FIRST_COL
 - GDSL types., 247
- GDSL_LOCATION_FIRST_ROW
 - GDSL types., 247
- GDSL_LOCATION_HEAD
 - GDSL types., 247
- GDSL_LOCATION_LAST
 - GDSL types., 247
- GDSL_LOCATION_LAST_COL
 - GDSL types., 247
- GDSL_LOCATION_LAST_ROW
 - GDSL types., 247
- GDSL_LOCATION_LEAF
 - GDSL types., 247
- GDSL_LOCATION_ROOT
 - GDSL types., 247
- GDSL_LOCATION_TAIL
 - GDSL types., 247
- GDSL_LOCATION_TOP
 - GDSL types., 247
- GDSL_LOCATION_UNDEF
 - GDSL types., 247
- GDSL_MAP_CONT
 - GDSL types., 247
- GDSL_MAP_STOP
 - GDSL types., 247
- GDSL_MAX
 - Various macros module., 178
- GDSL_MIN
 - Various macros module., 178
- GDSL_PERM_POSITION_FIRST
 - Permutation manipulation module., 182
- GDSL_PERM_POSITION_LAST
 - Permutation manipulation module., 182
- Hashtable manipulation module., 99
 - gds_hash, 101
 - gds_hash_alloc, 102
 - gds_hash_delete, 110
 - gds_hash_dump, 114
 - gds_hash_flush, 103
 - gds_hash_free, 103
 - gds_hash_func_t, 101
 - gds_hash_get_entries_number, 105
 - gds_hash_get_fill_factor, 107
 - gds_hash_get_lists_max_size, 105
 - gds_hash_get_longest_list_size, 106
 - gds_hash_get_name, 104
 - gds_hash_get_size, 106
 - gds_hash_insert, 108
 - gds_hash_map, 112
 - gds_hash_modify, 111
 - gds_hash_remove, 109
 - gds_hash_search, 111
 - gds_hash_set_name, 108

- gdsl_hash_t, 100
- gdsl_hash_write, 113
- gdsl_hash_write_xml, 114
- gdsl_key_func_t, 100
- Heap manipulation module., 116
 - gdsl_heap_alloc, 117
 - gdsl_heap_delete_top, 124
 - gdsl_heap_dump, 127
 - gdsl_heap_flush, 118
 - gdsl_heap_free, 118
 - gdsl_heap_get_name, 119
 - gdsl_heap_get_size, 120
 - gdsl_heap_get_top, 120
 - gdsl_heap_insert, 123
 - gdsl_heap_is_empty, 121
 - gdsl_heap_map_forward, 125
 - gdsl_heap_remove_top, 123
 - gdsl_heap_set_name, 121
 - gdsl_heap_set_top, 122
 - gdsl_heap_t, 117
 - gdsl_heap_write, 125
 - gdsl_heap_write_xml, 126
- Interval Heap manipulation module., 128
 - gdsl_interval_heap_alloc, 129
 - gdsl_interval_heap_delete_max, 138
 - gdsl_interval_heap_delete_min, 137
 - gdsl_interval_heap_dump, 141
 - gdsl_interval_heap_flush, 131
 - gdsl_interval_heap_free, 130
 - gdsl_interval_heap_get_max, 137
 - gdsl_interval_heap_get_min, 136
 - gdsl_interval_heap_get_name, 132
 - gdsl_interval_heap_get_size, 132
 - gdsl_interval_heap_insert, 134
 - gdsl_interval_heap_is_empty, 133
 - gdsl_interval_heap_map_forward, 139
 - gdsl_interval_heap_remove_max, 135
 - gdsl_interval_heap_remove_min, 136
 - gdsl_interval_heap_set_max_size, 133
 - gdsl_interval_heap_set_name, 134
 - gdsl_interval_heap_t, 129
 - gdsl_interval_heap_write, 139
 - gdsl_interval_heap_write_xml, 140
- Low level binary tree manipulation module., 15
 - _gdsl_bintree_alloc, 18
 - _gdsl_bintree_copy, 19
 - _gdsl_bintree_dump, 35
 - _gdsl_bintree_free, 19
 - _gdsl_bintree_get_content, 22
 - _gdsl_bintree_get_height, 25
 - _gdsl_bintree_get_left, 23
 - _gdsl_bintree_get_left_ref, 24
 - _gdsl_bintree_get_parent, 22
 - _gdsl_bintree_get_right, 23
 - _gdsl_bintree_get_right_ref, 25
 - _gdsl_bintree_get_size, 26
 - _gdsl_bintree_is_empty, 20
 - _gdsl_bintree_is_leaf, 21
 - _gdsl_bintree_is_root, 21
 - _gdsl_bintree_map_func_t, 17
 - _gdsl_bintree_map_infix, 32
 - _gdsl_bintree_map_postfix, 32
 - _gdsl_bintree_map_prefix, 31
 - _gdsl_bintree_rotate_left, 28
 - _gdsl_bintree_rotate_left_right, 30
 - _gdsl_bintree_rotate_right, 29
 - _gdsl_bintree_rotate_right_left, 30
 - _gdsl_bintree_set_content, 26
 - _gdsl_bintree_set_left, 27
 - _gdsl_bintree_set_parent, 27
 - _gdsl_bintree_set_right, 28
 - _gdsl_bintree_t, 17
 - _gdsl_bintree_write, 33
 - _gdsl_bintree_write_func_t, 17
 - _gdsl_bintree_write_xml, 34
- Low-level binary search tree manipulation module., 36
 - _gdsl_bstree_alloc, 38
 - _gdsl_bstree_copy, 39
 - _gdsl_bstree_dump, 52
 - _gdsl_bstree_free, 39
 - _gdsl_bstree_get_content, 41
 - _gdsl_bstree_get_height, 45
 - _gdsl_bstree_get_left, 43
 - _gdsl_bstree_get_parent, 42
 - _gdsl_bstree_get_right, 43
 - _gdsl_bstree_get_size, 44
 - _gdsl_bstree_insert, 45
 - _gdsl_bstree_is_empty, 40
 - _gdsl_bstree_is_leaf, 41
 - _gdsl_bstree_is_root, 42
 - _gdsl_bstree_map_func_t, 37
 - _gdsl_bstree_map_infix, 49
 - _gdsl_bstree_map_postfix, 49
 - _gdsl_bstree_map_prefix, 48

- `_gdsl_bstree_remove`, 46
- `_gdsl_bstree_search`, 47
- `_gdsl_bstree_search_next`, 47
- `_gdsl_bstree_t`, 37
- `_gdsl_bstree_write`, 50
- `_gdsl_bstree_write_func_t`, 38
- `_gdsl_bstree_write_xml`, 51
- Low-level doubly-linked list manipulation
 - module., 53
 - `_gdsl_list_alloc`, 54
 - `_gdsl_list_dump`, 62
 - `_gdsl_list_free`, 54
 - `_gdsl_list_get_size`, 56
 - `_gdsl_list_insert_after`, 56
 - `_gdsl_list_insert_before`, 57
 - `_gdsl_list_is_empty`, 55
 - `_gdsl_list_link`, 56
 - `_gdsl_list_map_backward`, 59
 - `_gdsl_list_map_forward`, 59
 - `_gdsl_list_remove`, 57
 - `_gdsl_list_search`, 58
 - `_gdsl_list_t`, 54
 - `_gdsl_list_write`, 60
 - `_gdsl_list_write_xml`, 61
- Low-level doubly-linked node manipulation
 - module., 63
 - `_gdsl_node_alloc`, 65
 - `_gdsl_node_dump`, 71
 - `_gdsl_node_free`, 65
 - `_gdsl_node_get_content`, 67
 - `_gdsl_node_get_pred`, 66
 - `_gdsl_node_get_succ`, 66
 - `_gdsl_node_link`, 69
 - `_gdsl_node_map_func_t`, 64
 - `_gdsl_node_set_content`, 68
 - `_gdsl_node_set_pred`, 68
 - `_gdsl_node_set_succ`, 67
 - `_gdsl_node_t`, 64
 - `_gdsl_node_unlink`, 69
 - `_gdsl_node_write`, 70
 - `_gdsl_node_write_func_t`, 64
 - `_gdsl_node_write_xml`, 71
- Main module, 73
 - `gdsl_get_version`, 73
- Permutation manipulation module., 180
 - `GDSL_PERM_POSITION_FIRST`, 182
 - `GDSL_PERM_POSITION_LAST`, 182
 - `gdsl_perm_alloc`, 182
 - `gdsl_perm_apply_on_array`, 195
 - `gdsl_perm_canonical_cycles_count`, 188
 - `gdsl_perm_canonical_to_linear`, 192
 - `gdsl_perm_copy`, 184
 - `gdsl_perm_data_t`, 182
 - `gdsl_perm_dump`, 197
 - `gdsl_perm_free`, 183
 - `gdsl_perm_get_element`, 186
 - `gdsl_perm_get_elements_array`, 186
 - `gdsl_perm_get_name`, 184
 - `gdsl_perm_get_size`, 185
 - `gdsl_perm_inverse`, 193
 - `gdsl_perm_linear_cycles_count`, 187
 - `gdsl_perm_linear_inversions_count`, 187
 - `gdsl_perm_linear_next`, 189
 - `gdsl_perm_linear_prev`, 190
 - `gdsl_perm_linear_to_canonical`, 192
 - `gdsl_perm_multiply`, 191
 - `gdsl_perm_position_t`, 182
 - `gdsl_perm_randomize`, 194
 - `gdsl_perm_reverse`, 194
 - `gdsl_perm_set_elements_array`, 190
 - `gdsl_perm_set_name`, 189
 - `gdsl_perm_t`, 182
 - `gdsl_perm_write`, 195
 - `gdsl_perm_write_func_t`, 182
 - `gdsl_perm_write_xml`, 196
- Queue manipulation module., 198
 - `gdsl_queue_alloc`, 199
 - `gdsl_queue_dump`, 210
 - `gdsl_queue_flush`, 201
 - `gdsl_queue_free`, 200
 - `gdsl_queue_get_head`, 203
 - `gdsl_queue_get_name`, 201
 - `gdsl_queue_get_size`, 202
 - `gdsl_queue_get_tail`, 203
 - `gdsl_queue_insert`, 205
 - `gdsl_queue_is_empty`, 202
 - `gdsl_queue_map_backward`, 208
 - `gdsl_queue_map_forward`, 207
 - `gdsl_queue_remove`, 205
 - `gdsl_queue_search`, 206
 - `gdsl_queue_search_by_position`, 207
 - `gdsl_queue_set_name`, 204
 - `gdsl_queue_t`, 199
 - `gdsl_queue_write`, 209
 - `gdsl_queue_write_xml`, 209

- Red-black tree manipulation module., 212
 - gdsl_rbtrees_alloc, 213
 - gdsl_rbtrees_delete, 220
 - gdsl_rbtrees_dump, 226
 - gdsl_rbtrees_flush, 215
 - gdsl_rbtrees_free, 214
 - gdsl_rbtrees_get_name, 215
 - gdsl_rbtrees_get_root, 216
 - gdsl_rbtrees_get_size, 217
 - gdsl_rbtrees_height, 217
 - gdsl_rbtrees_insert, 219
 - gdsl_rbtrees_is_empty, 216
 - gdsl_rbtrees_map_infix, 222
 - gdsl_rbtrees_map_postfix, 223
 - gdsl_rbtrees_map_prefix, 222
 - gdsl_rbtrees_remove, 219
 - gdsl_rbtrees_search, 221
 - gdsl_rbtrees_set_name, 218
 - gdsl_rbtrees_t, 213
 - gdsl_rbtrees_write, 224
 - gdsl_rbtrees_write_xml, 225
- Sort module., 227
 - gdsl_sort, 227
- Stack manipulation module., 228
 - gdsl_stack_alloc, 229
 - gdsl_stack_dump, 242
 - gdsl_stack_flush, 231
 - gdsl_stack_free, 230
 - gdsl_stack_get_bottom, 234
 - gdsl_stack_get_growing_factor, 232
 - gdsl_stack_get_name, 231
 - gdsl_stack_get_size, 232
 - gdsl_stack_get_top, 234
 - gdsl_stack_insert, 236
 - gdsl_stack_is_empty, 233
 - gdsl_stack_map_backward, 240
 - gdsl_stack_map_forward, 239
 - gdsl_stack_remove, 237
 - gdsl_stack_search, 238
 - gdsl_stack_search_by_position, 238
 - gdsl_stack_set_growing_factor, 235
 - gdsl_stack_set_name, 235
 - gdsl_stack_t, 229
 - gdsl_stack_write, 240
 - gdsl_stack_write_xml, 241
- TRUE
 - GDSL types., 247
- Various macros module., 178
 - GDSL_MAX, 178
 - GDSL_MIN, 178
- _gdsl_bintree.h, 249
- _gdsl_bintree_alloc
 - Low level binary tree manipulation module., 18
- _gdsl_bintree_copy
 - Low level binary tree manipulation module., 19
- _gdsl_bintree_dump
 - Low level binary tree manipulation module., 35
- _gdsl_bintree_free
 - Low level binary tree manipulation module., 19
- _gdsl_bintree_get_content
 - Low level binary tree manipulation module., 22
- _gdsl_bintree_get_height
 - Low level binary tree manipulation module., 25
- _gdsl_bintree_get_left
 - Low level binary tree manipulation module., 23
- _gdsl_bintree_get_left_ref
 - Low level binary tree manipulation module., 24
- _gdsl_bintree_get_parent
 - Low level binary tree manipulation module., 22
- _gdsl_bintree_get_right
 - Low level binary tree manipulation module., 23
- _gdsl_bintree_get_right_ref
 - Low level binary tree manipulation module., 25
- _gdsl_bintree_get_size
 - Low level binary tree manipulation module., 26
- _gdsl_bintree_is_empty
 - Low level binary tree manipulation module., 20
- _gdsl_bintree_is_leaf
 - Low level binary tree manipulation module., 21
- _gdsl_bintree_is_root
 - Low level binary tree manipulation module., 21
- _gdsl_bintree_map_func_t
 - Low level binary tree manipulation module., 17
- _gdsl_bintree_map_infix

- Low level binary tree manipulation module., 32
- `_gdsl_bintree_map_postfix`
 - Low level binary tree manipulation module., 32
- `_gdsl_bintree_map_prefix`
 - Low level binary tree manipulation module., 31
- `_gdsl_bintree_rotate_left`
 - Low level binary tree manipulation module., 28
- `_gdsl_bintree_rotate_left_right`
 - Low level binary tree manipulation module., 30
- `_gdsl_bintree_rotate_right`
 - Low level binary tree manipulation module., 29
- `_gdsl_bintree_rotate_right_left`
 - Low level binary tree manipulation module., 30
- `_gdsl_bintree_set_content`
 - Low level binary tree manipulation module., 26
- `_gdsl_bintree_set_left`
 - Low level binary tree manipulation module., 27
- `_gdsl_bintree_set_parent`
 - Low level binary tree manipulation module., 27
- `_gdsl_bintree_set_right`
 - Low level binary tree manipulation module., 28
- `_gdsl_bintree_t`
 - Low level binary tree manipulation module., 17
- `_gdsl_bintree_write`
 - Low level binary tree manipulation module., 33
- `_gdsl_bintree_write_func_t`
 - Low level binary tree manipulation module., 17
- `_gdsl_bintree_write_xml`
 - Low level binary tree manipulation module., 34
- `_gdsl_bstree.h`, 251
- `_gdsl_bstree_alloc`
 - Low-level binary search tree manipulation module., 38
- `_gdsl_bstree_copy`
 - Low-level binary search tree manipulation module., 39
- `_gdsl_bstree_dump`
 - Low-level binary search tree manipulation module., 52
- `_gdsl_bstree_free`
 - Low-level binary search tree manipulation module., 39
- `_gdsl_bstree_get_content`
 - Low-level binary search tree manipulation module., 41
- `_gdsl_bstree_get_height`
 - Low-level binary search tree manipulation module., 45
- `_gdsl_bstree_get_left`
 - Low-level binary search tree manipulation module., 43
- `_gdsl_bstree_get_parent`
 - Low-level binary search tree manipulation module., 42
- `_gdsl_bstree_get_right`
 - Low-level binary search tree manipulation module., 43
- `_gdsl_bstree_get_size`
 - Low-level binary search tree manipulation module., 44
- `_gdsl_bstree_insert`
 - Low-level binary search tree manipulation module., 45
- `_gdsl_bstree_is_empty`
 - Low-level binary search tree manipulation module., 40
- `_gdsl_bstree_is_leaf`
 - Low-level binary search tree manipulation module., 41
- `_gdsl_bstree_is_root`
 - Low-level binary search tree manipulation module., 42
- `_gdsl_bstree_map_func_t`
 - Low-level binary search tree manipulation module., 37
- `_gdsl_bstree_map_infix`
 - Low-level binary search tree manipulation module., 49
- `_gdsl_bstree_map_postfix`
 - Low-level binary search tree manipulation module., 49
- `_gdsl_bstree_map_prefix`
 - Low-level binary search tree manipulation module., 48

- `_gdsl_bstree_remove`
Low-level binary search tree manipulation module., 46
 - `_gdsl_bstree_search`
Low-level binary search tree manipulation module., 47
 - `_gdsl_bstree_search_next`
Low-level binary search tree manipulation module., 47
 - `_gdsl_bstree_t`
Low-level binary search tree manipulation module., 37
 - `_gdsl_bstree_write`
Low-level binary search tree manipulation module., 50
 - `_gdsl_bstree_write_func_t`
Low-level binary search tree manipulation module., 38
 - `_gdsl_bstree_write_xml`
Low-level binary search tree manipulation module., 51
- `_gdsl_list.h`, 253
- `_gdsl_list_alloc`
Low-level doubly-linked list manipulation module., 54
- `_gdsl_list_dump`
Low-level doubly-linked list manipulation module., 62
- `_gdsl_list_free`
Low-level doubly-linked list manipulation module., 54
- `_gdsl_list_get_size`
Low-level doubly-linked list manipulation module., 56
- `_gdsl_list_insert_after`
Low-level doubly-linked list manipulation module., 56
- `_gdsl_list_insert_before`
Low-level doubly-linked list manipulation module., 57
- `_gdsl_list_is_empty`
Low-level doubly-linked list manipulation module., 55
- `_gdsl_list_link`
Low-level doubly-linked list manipulation module., 56
- `_gdsl_list_map_backward`
Low-level doubly-linked list manipulation module., 59
- `_gdsl_list_map_forward`
Low-level doubly-linked list manipulation module., 59
- `_gdsl_list_remove`
Low-level doubly-linked list manipulation module., 57
- `_gdsl_list_search`
Low-level doubly-linked list manipulation module., 58
- `_gdsl_list_t`
Low-level doubly-linked list manipulation module., 54
- `_gdsl_list_write`
Low-level doubly-linked list manipulation module., 60
- `_gdsl_list_write_xml`
Low-level doubly-linked list manipulation module., 61
- `_gdsl_node.h`, 254
- `_gdsl_node_alloc`
Low-level doubly-linked node manipulation module., 65
- `_gdsl_node_dump`
Low-level doubly-linked node manipulation module., 71
- `_gdsl_node_free`
Low-level doubly-linked node manipulation module., 65
- `_gdsl_node_get_content`
Low-level doubly-linked node manipulation module., 67
- `_gdsl_node_get_pred`
Low-level doubly-linked node manipulation module., 66
- `_gdsl_node_get_succ`
Low-level doubly-linked node manipulation module., 66
- `_gdsl_node_link`
Low-level doubly-linked node manipulation module., 69
- `_gdsl_node_map_func_t`
Low-level doubly-linked node manipulation module., 64
- `_gdsl_node_set_content`
Low-level doubly-linked node manipulation module., 68
- `_gdsl_node_set_pred`
Low-level doubly-linked node manipulation module., 68
- `_gdsl_node_set_succ`

- Low-level doubly-linked node manipulation module., 67
- `_gdsl_node_t`
 - Low-level doubly-linked node manipulation module., 64
- `_gdsl_node_unlink`
 - Low-level doubly-linked node manipulation module., 69
- `_gdsl_node_write`
 - Low-level doubly-linked node manipulation module., 70
- `_gdsl_node_write_func_t`
 - Low-level doubly-linked node manipulation module., 64
- `_gdsl_node_write_xml`
 - Low-level doubly-linked node manipulation module., 71
- 2D-Arrays manipulation module., 74
 - `gdsl_2darray_alloc`, 75
 - `gdsl_2darray_dump`, 82
 - `gdsl_2darray_free`, 76
 - `gdsl_2darray_get_columns_number`, 78
 - `gdsl_2darray_get_content`, 79
 - `gdsl_2darray_get_name`, 76
 - `gdsl_2darray_get_rows_number`, 77
 - `gdsl_2darray_get_size`, 78
 - `gdsl_2darray_set_content`, 80
 - `gdsl_2darray_set_name`, 80
 - `gdsl_2darray_t`, 75
 - `gdsl_2darray_write`, 81
 - `gdsl_2darray_write_xml`, 82
- `bool`
 - GDSL types., 247
- `gdsl.h`, 255
- `gdsl_2darray.h`, 255
- `gdsl_2darray_alloc`
 - 2D-Arrays manipulation module., 75
- `gdsl_2darray_dump`
 - 2D-Arrays manipulation module., 82
- `gdsl_2darray_free`
 - 2D-Arrays manipulation module., 76
- `gdsl_2darray_get_columns_number`
 - 2D-Arrays manipulation module., 78
- `gdsl_2darray_get_content`
 - 2D-Arrays manipulation module., 79
- `gdsl_2darray_get_name`
 - 2D-Arrays manipulation module., 76
- `gdsl_2darray_get_rows_number`
 - 2D-Arrays manipulation module., 77
- `gdsl_2darray_get_size`
 - 2D-Arrays manipulation module., 78
- `gdsl_2darray_set_content`
 - 2D-Arrays manipulation module., 80
- `gdsl_2darray_set_name`
 - 2D-Arrays manipulation module., 80
- `gdsl_2darray_t`
 - 2D-Arrays manipulation module., 75
- `gdsl_2darray_write`
 - 2D-Arrays manipulation module., 81
- `gdsl_2darray_write_xml`
 - 2D-Arrays manipulation module., 82
- `gdsl_alloc_func_t`
 - GDSL types., 244
- `gdsl_bstree.h`, 257
- `gdsl_bstree_alloc`
 - Binary search tree manipulation module., 85
- `gdsl_bstree_delete`
 - Binary search tree manipulation module., 92
- `gdsl_bstree_dump`
 - Binary search tree manipulation module., 98
- `gdsl_bstree_flush`
 - Binary search tree manipulation module., 87
- `gdsl_bstree_free`
 - Binary search tree manipulation module., 86
- `gdsl_bstree_get_height`
 - Binary search tree manipulation module., 90
- `gdsl_bstree_get_name`
 - Binary search tree manipulation module., 87
- `gdsl_bstree_get_root`
 - Binary search tree manipulation module., 89
- `gdsl_bstree_get_size`
 - Binary search tree manipulation module., 89
- `gdsl_bstree_insert`
 - Binary search tree manipulation module., 91
- `gdsl_bstree_is_empty`
 - Binary search tree manipulation module., 88

- gdsl_bstree_map_infix
 - Binary search tree manipulation module., 95
- gdsl_bstree_map_postfix
 - Binary search tree manipulation module., 95
- gdsl_bstree_map_prefix
 - Binary search tree manipulation module., 94
- gdsl_bstree_remove
 - Binary search tree manipulation module., 92
- gdsl_bstree_search
 - Binary search tree manipulation module., 93
- gdsl_bstree_set_name
 - Binary search tree manipulation module., 90
- gdsl_bstree_t
 - Binary search tree manipulation module., 85
- gdsl_bstree_write
 - Binary search tree manipulation module., 96
- gdsl_bstree_write_xml
 - Binary search tree manipulation module., 97
- gdsl_compare_func_t
 - GDSL types., 245
- gdsl_constant_t
 - GDSL types., 246
- gdsl_copy_func_t
 - GDSL types., 245
- gdsl_element_t
 - GDSL types., 244
- gdsl_free_func_t
 - GDSL types., 244
- gdsl_get_version
 - Main module, 73
- gdsl_hash
 - Hashtable manipulation module., 101
- gdsl_hash.h, 258
- gdsl_hash_alloc
 - Hashtable manipulation module., 102
- gdsl_hash_delete
 - Hashtable manipulation module., 110
- gdsl_hash_dump
 - Hashtable manipulation module., 114
- gdsl_hash_flush
 - Hashtable manipulation module., 103
- gdsl_hash_free
 - Hashtable manipulation module., 103
- gdsl_hash_func_t
 - Hashtable manipulation module., 101
- gdsl_hash_get_entries_number
 - Hashtable manipulation module., 105
- gdsl_hash_get_fill_factor
 - Hashtable manipulation module., 107
- gdsl_hash_get_lists_max_size
 - Hashtable manipulation module., 105
- gdsl_hash_get_longest_list_size
 - Hashtable manipulation module., 106
- gdsl_hash_get_name
 - Hashtable manipulation module., 104
- gdsl_hash_get_size
 - Hashtable manipulation module., 106
- gdsl_hash_insert
 - Hashtable manipulation module., 108
- gdsl_hash_map
 - Hashtable manipulation module., 112
- gdsl_hash_modify
 - Hashtable manipulation module., 111
- gdsl_hash_remove
 - Hashtable manipulation module., 109
- gdsl_hash_search
 - Hashtable manipulation module., 111
- gdsl_hash_set_name
 - Hashtable manipulation module., 108
- gdsl_hash_t
 - Hashtable manipulation module., 100
- gdsl_hash_write
 - Hashtable manipulation module., 113
- gdsl_hash_write_xml
 - Hashtable manipulation module., 114
- gdsl_heap.h, 260
- gdsl_heap_alloc
 - Heap manipulation module., 117
- gdsl_heap_delete_top
 - Heap manipulation module., 124
- gdsl_heap_dump
 - Heap manipulation module., 127
- gdsl_heap_flush
 - Heap manipulation module., 118
- gdsl_heap_free
 - Heap manipulation module., 118
- gdsl_heap_get_name
 - Heap manipulation module., 119
- gdsl_heap_get_size
 - Heap manipulation module., 120
- gdsl_heap_get_top

- Heap manipulation module., 120
- gdsl_heap_insert
 - Heap manipulation module., 123
- gdsl_heap_is_empty
 - Heap manipulation module., 121
- gdsl_heap_map_forward
 - Heap manipulation module., 125
- gdsl_heap_remove_top
 - Heap manipulation module., 123
- gdsl_heap_set_name
 - Heap manipulation module., 121
- gdsl_heap_set_top
 - Heap manipulation module., 122
- gdsl_heap_t
 - Heap manipulation module., 117
- gdsl_heap_write
 - Heap manipulation module., 125
- gdsl_heap_write_xml
 - Heap manipulation module., 126
- gdsl_interval_heap.h, 261
- gdsl_interval_heap_alloc
 - Interval Heap manipulation module., 129
- gdsl_interval_heap_delete_max
 - Interval Heap manipulation module., 138
- gdsl_interval_heap_delete_min
 - Interval Heap manipulation module., 137
- gdsl_interval_heap_dump
 - Interval Heap manipulation module., 141
- gdsl_interval_heap_flush
 - Interval Heap manipulation module., 131
- gdsl_interval_heap_free
 - Interval Heap manipulation module., 130
- gdsl_interval_heap_get_max
 - Interval Heap manipulation module., 137
- gdsl_interval_heap_get_min
 - Interval Heap manipulation module., 136
- gdsl_interval_heap_get_name
 - Interval Heap manipulation module., 132
- gdsl_interval_heap_get_size
 - Interval Heap manipulation module., 132
- gdsl_interval_heap_insert
 - Interval Heap manipulation module., 134
- gdsl_interval_heap_is_empty
 - Interval Heap manipulation module., 133
- gdsl_interval_heap_map_forward
 - Interval Heap manipulation module., 139
- gdsl_interval_heap_remove_max
 - Interval Heap manipulation module., 135
- gdsl_interval_heap_remove_min
 - Interval Heap manipulation module., 136
- gdsl_interval_heap_set_max_size
 - Interval Heap manipulation module., 133
- gdsl_interval_heap_set_name
 - Interval Heap manipulation module., 134
- gdsl_interval_heap_t
 - Interval Heap manipulation module., 129
- gdsl_interval_heap_write
 - Interval Heap manipulation module., 139
- gdsl_interval_heap_write_xml
 - Interval Heap manipulation module., 140
- gdsl_key_func_t
 - Hashtable manipulation module., 100
- gdsl_list.h, 263
- gdsl_list_alloc
 - Doubly-linked list manipulation module., 145
- gdsl_list_cursor_alloc
 - Doubly-linked list manipulation module., 163
- gdsl_list_cursor_delete
 - Doubly-linked list manipulation module., 175
- gdsl_list_cursor_delete_after
 - Doubly-linked list manipulation module., 175
- gdsl_list_cursor_delete_before
 - Doubly-linked list manipulation module., 176
- gdsl_list_cursor_free

- Doubly-linked list manipulation module., 164
- gdsl_list_cursor_get_content
 - Doubly-linked list manipulation module., 171
- gdsl_list_cursor_has_pred
 - Doubly-linked list manipulation module., 170
- gdsl_list_cursor_has_succ
 - Doubly-linked list manipulation module., 169
- gdsl_list_cursor_insert_after
 - Doubly-linked list manipulation module., 171
- gdsl_list_cursor_insert_before
 - Doubly-linked list manipulation module., 172
- gdsl_list_cursor_is_on_head
 - Doubly-linked list manipulation module., 168
- gdsl_list_cursor_is_on_tail
 - Doubly-linked list manipulation module., 168
- gdsl_list_cursor_move_to_head
 - Doubly-linked list manipulation module., 164
- gdsl_list_cursor_move_to_position
 - Doubly-linked list manipulation module., 166
- gdsl_list_cursor_move_to_tail
 - Doubly-linked list manipulation module., 165
- gdsl_list_cursor_move_to_value
 - Doubly-linked list manipulation module., 165
- gdsl_list_cursor_remove
 - Doubly-linked list manipulation module., 173
- gdsl_list_cursor_remove_after
 - Doubly-linked list manipulation module., 173
- gdsl_list_cursor_remove_before
 - Doubly-linked list manipulation module., 174
- gdsl_list_cursor_set_content
 - Doubly-linked list manipulation module., 170
- gdsl_list_cursor_step_backward
 - Doubly-linked list manipulation module., 167
- gdsl_list_cursor_step_forward
 - Doubly-linked list manipulation module., 167
- gdsl_list_cursor_t
 - Doubly-linked list manipulation module., 145
- gdsl_list_delete
 - Doubly-linked list manipulation module., 155
- gdsl_list_delete_head
 - Doubly-linked list manipulation module., 154
- gdsl_list_delete_tail
 - Doubly-linked list manipulation module., 155
- gdsl_list_dump
 - Doubly-linked list manipulation module., 163
- gdsl_list_flush
 - Doubly-linked list manipulation module., 146
- gdsl_list_free
 - Doubly-linked list manipulation module., 146
- gdsl_list_get_head
 - Doubly-linked list manipulation module., 149
- gdsl_list_get_name
 - Doubly-linked list manipulation module., 147
- gdsl_list_get_size
 - Doubly-linked list manipulation module., 148
- gdsl_list_get_tail
 - Doubly-linked list manipulation module., 149
- gdsl_list_insert_head
 - Doubly-linked list manipulation module., 150
- gdsl_list_insert_tail
 - Doubly-linked list manipulation module., 151
- gdsl_list_is_empty
 - Doubly-linked list manipulation module., 148
- gdsl_list_map_backward
 - Doubly-linked list manipulation module., 160
- gdsl_list_map_forward

- Doubly-linked list manipulation module., 160
- gdsl_list_remove
 - Doubly-linked list manipulation module., 153
- gdsl_list_remove_head
 - Doubly-linked list manipulation module., 152
- gdsl_list_remove_tail
 - Doubly-linked list manipulation module., 153
- gdsl_list_search
 - Doubly-linked list manipulation module., 156
- gdsl_list_search_by_position
 - Doubly-linked list manipulation module., 157
- gdsl_list_search_max
 - Doubly-linked list manipulation module., 158
- gdsl_list_search_min
 - Doubly-linked list manipulation module., 158
- gdsl_list_set_name
 - Doubly-linked list manipulation module., 150
- gdsl_list_sort
 - Doubly-linked list manipulation module., 159
- gdsl_list_t
 - Doubly-linked list manipulation module., 145
- gdsl_list_write
 - Doubly-linked list manipulation module., 161
- gdsl_list_write_xml
 - Doubly-linked list manipulation module., 162
- gdsl_location_t
 - GDSL types., 247
- gdsl_macros.h, 266
- gdsl_map_func_t
 - GDSL types., 245
- gdsl_perm.h, 266
- gdsl_perm_alloc
 - Permutation manipulation module., 182
- gdsl_perm_apply_on_array
 - Permutation manipulation module., 195
- gdsl_perm_canonical_cycles_count
 - Permutation manipulation module., 188
- gdsl_perm_canonical_to_linear
 - Permutation manipulation module., 192
- gdsl_perm_copy
 - Permutation manipulation module., 184
- gdsl_perm_data_t
 - Permutation manipulation module., 182
- gdsl_perm_dump
 - Permutation manipulation module., 197
- gdsl_perm_free
 - Permutation manipulation module., 183
- gdsl_perm_get_element
 - Permutation manipulation module., 186
- gdsl_perm_get_elements_array
 - Permutation manipulation module., 186
- gdsl_perm_get_name
 - Permutation manipulation module., 184
- gdsl_perm_get_size
 - Permutation manipulation module., 185
- gdsl_perm_inverse
 - Permutation manipulation module., 193
- gdsl_perm_linear_cycles_count
 - Permutation manipulation module., 187
- gdsl_perm_linear_inversions_count
 - Permutation manipulation module., 187
- gdsl_perm_linear_next
 - Permutation manipulation module., 189
- gdsl_perm_linear_prev
 - Permutation manipulation module., 190
- gdsl_perm_linear_to_canonical
 - Permutation manipulation module., 192
- gdsl_perm_multiply

- Permutation manipulation module., 191
- gdsl_perm_position_t
 - Permutation manipulation module., 182
- gdsl_perm_randomize
 - Permutation manipulation module., 194
- gdsl_perm_reverse
 - Permutation manipulation module., 194
- gdsl_perm_set_elements_array
 - Permutation manipulation module., 190
- gdsl_perm_set_name
 - Permutation manipulation module., 189
- gdsl_perm_t
 - Permutation manipulation module., 182
- gdsl_perm_write
 - Permutation manipulation module., 195
- gdsl_perm_write_func_t
 - Permutation manipulation module., 182
- gdsl_perm_write_xml
 - Permutation manipulation module., 196
- gdsl_queue.h, 268
- gdsl_queue_alloc
 - Queue manipulation module., 199
- gdsl_queue_dump
 - Queue manipulation module., 210
- gdsl_queue_flush
 - Queue manipulation module., 201
- gdsl_queue_free
 - Queue manipulation module., 200
- gdsl_queue_get_head
 - Queue manipulation module., 203
- gdsl_queue_get_name
 - Queue manipulation module., 201
- gdsl_queue_get_size
 - Queue manipulation module., 202
- gdsl_queue_get_tail
 - Queue manipulation module., 203
- gdsl_queue_insert
 - Queue manipulation module., 205
- gdsl_queue_is_empty
 - Queue manipulation module., 202
- gdsl_queue_map_backward
 - Queue manipulation module., 208
- gdsl_queue_map_forward
 - Queue manipulation module., 207
- gdsl_queue_remove
 - Queue manipulation module., 205
- gdsl_queue_search
 - Queue manipulation module., 206
- gdsl_queue_search_by_position
 - Queue manipulation module., 207
- gdsl_queue_set_name
 - Queue manipulation module., 204
- gdsl_queue_t
 - Queue manipulation module., 199
- gdsl_queue_write
 - Queue manipulation module., 209
- gdsl_queue_write_xml
 - Queue manipulation module., 209
- gdsl_rbtrees.h, 269
- gdsl_rbtrees_alloc
 - Red-black tree manipulation module., 213
- gdsl_rbtrees_delete
 - Red-black tree manipulation module., 220
- gdsl_rbtrees_dump
 - Red-black tree manipulation module., 226
- gdsl_rbtrees_flush
 - Red-black tree manipulation module., 215
- gdsl_rbtrees_free
 - Red-black tree manipulation module., 214
- gdsl_rbtrees_get_name
 - Red-black tree manipulation module., 215
- gdsl_rbtrees_get_root
 - Red-black tree manipulation module., 216
- gdsl_rbtrees_get_size
 - Red-black tree manipulation module., 217
- gdsl_rbtrees_height
 - Red-black tree manipulation module., 217
- gdsl_rbtrees_insert
 - Red-black tree manipulation module., 219
- gdsl_rbtrees_is_empty

- Red-black tree manipulation module., 216
- gds_l_rbtrees_map_infix
 - Red-black tree manipulation module., 222
- gds_l_rbtrees_map_postfix
 - Red-black tree manipulation module., 223
- gds_l_rbtrees_map_prefix
 - Red-black tree manipulation module., 222
- gds_l_rbtrees_remove
 - Red-black tree manipulation module., 219
- gds_l_rbtrees_search
 - Red-black tree manipulation module., 221
- gds_l_rbtrees_set_name
 - Red-black tree manipulation module., 218
- gds_l_rbtrees_t
 - Red-black tree manipulation module., 213
- gds_l_rbtrees_write
 - Red-black tree manipulation module., 224
- gds_l_rbtrees_write_xml
 - Red-black tree manipulation module., 225
- gds_l_sort
 - Sort module., 227
- gds_l_sort.h, 271
- gds_l_stack.h, 271
- gds_l_stack_alloc
 - Stack manipulation module., 229
- gds_l_stack_dump
 - Stack manipulation module., 242
- gds_l_stack_flush
 - Stack manipulation module., 231
- gds_l_stack_free
 - Stack manipulation module., 230
- gds_l_stack_get_bottom
 - Stack manipulation module., 234
- gds_l_stack_get_growing_factor
 - Stack manipulation module., 232
- gds_l_stack_get_name
 - Stack manipulation module., 231
- gds_l_stack_get_size
 - Stack manipulation module., 232
- gds_l_stack_get_top
 - Stack manipulation module., 234
- gds_l_stack_insert
 - Stack manipulation module., 236
- gds_l_stack_is_empty
 - Stack manipulation module., 233
- gds_l_stack_map_backward
 - Stack manipulation module., 240
- gds_l_stack_map_forward
 - Stack manipulation module., 239
- gds_l_stack_remove
 - Stack manipulation module., 237
- gds_l_stack_search
 - Stack manipulation module., 238
- gds_l_stack_search_by_position
 - Stack manipulation module., 238
- gds_l_stack_set_growing_factor
 - Stack manipulation module., 235
- gds_l_stack_set_name
 - Stack manipulation module., 235
- gds_l_stack_t
 - Stack manipulation module., 229
- gds_l_stack_write
 - Stack manipulation module., 240
- gds_l_stack_write_xml
 - Stack manipulation module., 241
- gds_l_types.h, 273
- gds_l_write_func_t
 - GDSL types., 246
- mainpage.h, 274
- ulong
 - GDSL types., 246
- ushort
 - GDSL types., 246