

multithreaded/p_stack.c

```
1  /*
2  A dynamic stack holding pointers
3  - note: this current implementation only resizes stack to increase size, never to decrease size
4  */
5
6  #include "p_stack.h"
7
8  /**
9   * @brief initialize stack with an initial size (capacity)
10   * @param p PSTACK*: a pointer to uninitialized memory
11   * @param stack_size size_t: initial capacity of the stack to be initialized
12   * @return 0 on success; 1 otherwise
13   */
14  int init_pstack(PSTACK *p, size_t stack_size)
15  {
16      if (p == NULL || stack_size == 0)
17      {
18          return 1;
19      }
20
21      p->size = stack_size;
22      p->pos = -1;
23      p->items = (void **)malloc(stack_size * sizeof(void *));
24
25      memset(p->items, 0, stack_size * sizeof(void *));
26
27      return 0;
28  }
29
30  /**
31   * @brief push an item onto the stack; if the stack is full, resize the stack
32   * @param p PSTACK*: (pointer to) the stack the function will push item onto
33   * @param item void*: pointer to push onto stack
34   * @return 0 on success; 1 otherwise
35   */
36  int push_pstack(PSTACK *p, void *item)
37  {
38      if (p == NULL)
39      {
40          return 1;
41      }
42
43      if (is_full_pstack(p))
44      {
45          resize_pstack(p);
46      }
47
48      ++(p->pos);
```

```
49     p->items[p->pos] = item;
50
51     return 0;
52 }
53
54 /**
55  * @brief pop from the stack
56  * @param p PSTACK*: (pointer to) the stack the function will pop from
57  * @param p_item void**: pointer that will be populated with popped element (which itself is a
58  * @return 0 on success; 1 otherwise
59  */
60 int pop_pstack(PSTACK *p, void **p_item)
61 {
62     if (p == NULL || is_empty_pstack(p))
63     {
64         return 1;
65     }
66
67     *p_item = p->items[p->pos];
68     p->items[p->pos] = NULL;
69     (p->pos)--;
70     return 0;
71 }
72
73 /**
74  * @brief check if the stack is full
75  * @param p PSTACK*: (pointer to) the stack to check
76  * @return true if full; false otherwise
77  */
78 bool is_full_pstack(PSTACK *p)
79 {
80     if (p == NULL)
81     {
82         return 0;
83     }
84     return (p->pos == (p->size - 1));
85 }
86
87 /**
88  * @brief check if the stack is empty
89  * @param p PSTACK*: (pointer to) the stack to check
90  * @return true if empty; false otherwise
91  */
92 bool is_empty_pstack(PSTACK *p)
93 {
94     if (p == NULL)
95     {
96         return 0;
97     }
```

```
98     return (p->pos == -1);
99 }
100
101 /**
102  * @brief resize stack to have greater capacity; maintain existing elements
103  * @param p PSTACK*: (pointer to) the stack to resize
104  * @return 0 on success; 1 otherwise
105  */
106 int resize_pstack(PSTACK *p)
107 {
108     size_t old_size = p->size;
109     void **old_items = p->items;
110     p->size = (p->size) * PSTACK_RESIZE_FACTOR;
111     p->items = (void **)malloc((p->size) * sizeof(void *));
112     if (p->items == NULL)
113     {
114         return 1;
115     }
116
117     for (size_t i = 0; i < old_size; ++i)
118     {
119         p->items[i] = old_items[i];
120     }
121     for (size_t i = old_size; i < p->size; ++i)
122     {
123         p->items[i] = NULL;
124     }
125
126     free(old_items);
127     old_items = NULL;
128
129     return 0;
130 }
131
132 /**
133  * @brief returns number of elements currently in the stack
134  * @param p STACK*: (pointer to) the stack
135  * @return number of elements in the stack
136  */
137 size_t num_elements_pstack(PSTACK *p)
138 {
139     return p->pos + 1;
140 }
141
142 /**
143  * @brief deconstruct stack: free all allocated memory
144  * @param p PSTACK*: (pointer to) the stack to deconstruct
145  * @return 0 on success; 1 otherwise
146  */
147 int cleanup_pstack(PSTACK *p)
```

```
148 {
149     if (p == NULL || p->items == NULL)
150     {
151         return 0;
152     }
153     for (size_t i = 0; i < p->size; ++i)
154     {
155         if (p->items[i] != NULL)
156         {
157             free(p->items[i]);
158             p->items[i] = NULL;
159         }
160     }
161     free(p->items);
162     p->items = NULL;
163     return 0;
164 }
```