GCC Code Coverage Report

			Exec	Total	Coverage
Directory:	./				
File:	project/src/utils.c	Lines:	101	201	50.2 %
Date:	2021-03-15 14:28:04	Branches:	62	138	44.9 %

Line	Branch	Exec	Source
1			#include <stdlib.h>
2			#include <stdio.h>
3			#include <memory.h>
4			
5			#include "utils.h"
6			
7			int print_buffer(const Tasks *tasks) {
8			if (tasks == NULL || tasks->tasks_amount == 0) {
9			return -1;
10			}
11			
12			for (size_t k = 0; k < tasks->tasks_amount; k++) {
13			printf("priority: %zu number: %zu\n", tasks->buffer[k].priority, tasks->buffer[k].number);
14			for (size_t i = 0; i < 3; ++i) {
15			printf("%zu ", tasks->buffer[k].date[i]);
16			}
17			printf("\n");
18			printf("%s\n----\n", tasks->buffer[k].description);
19			}
20			
21			return 0;
22			}
23			
24		44	int date_comparator(const size_t *lhs, const size_t *rhs) {
25	✓✓✗✓	44	if (lhs == NULL || rhs == NULL) {
26		2	return -1;
27			}
28			
29	✓✗	110	for (int i = 2; i >= 0; --i) {
30	✓✓	110	if (lhs[i] > rhs[i]) {
31		10	return LHS_IS_LARGER;
32	✓✓	100	} else if (lhs[i] < rhs[i]) {
33		6	return RHS_IS_LARGER;
34			}
35	✓✓	94	if (!i) {
36		26	break;
37			}
38			}
39			
40		26	return EQUAL;
41			}
42			
43		32	int tasks_comparator(const Task *lhs, const Task *rhs) {
44	✓✗✗✓	32	if (lhs == NULL || rhs == NULL) {
45			return -1;
46			}
47			
48	✓✓	32	if (lhs->priority > rhs->priority) {
49		4	return LHS_IS_LARGER;
50	✓✓	28	} else if (lhs->priority < rhs->priority) {
51		12	return RHS_IS_LARGER;
52			}
53	✓✓	16	if (date_comparator(lhs->date, rhs->date) == LHS_IS_LARGER) {
54		4	return LHS_IS_LARGER;
55	✗✓	12	} else if (date_comparator(lhs->date, rhs->date) == RHS_IS_LARGER) {
56			return RHS_IS_LARGER;
57			}
58			
59		12	return EQUAL;
60			}
61			
62		8	int sort(Task *task, const size_t size) {
63	✓✗✗✓	8	if (task == NULL || task->number == 0) {
64			return -1;
65			}
66			

```
 67          8    size_t i = 0;
 68          8    size_t j = size - 1;
 69          8    Task mid = task[size / 2];
 70
 71    ✓✓   16    while (i <= j) {
 72    ✓✓   16        while (tasks_comparator(&task[i], &mid) == RHS_IS_LARGER) {
 73          8            i++;
 74                   }
 75
 76    ✓✓   12        while (tasks_comparator(&task[j], &mid) == LHS_IS_LARGER) {
 77          4            j--;
 78                   }
 79
 80 ✗✓✗✗    8        if (j == 0 && i == 0) {
 81                       return 0;
 82                   }
 83
 84    ✓✗    8        if (i <= j) {
 85          8            Task tmp = task[i];
 86          8            task[i] = task[j];
 87          8            task[j] = tmp;
 88
 89          8            i++;
 90          8            j--;
 91                   }
 92               }
 93
 94    ✓✓    8    if (j > 0) {
 95          2        sort(task, j + 1);
 96               }
 97    ✓✓    8    if (i < size) {
 98          4        sort(&task[i], size - i);
 99               }
100
101          8    return 0;
102           }
103
104         12int parse_date(const char *date_str, size_t *date_arr) {
105 ✓✓✗✓   12    if (date_str == NULL || date_arr == NULL) {
106          2        return -1;
107               }
108
109         10    int i = 0;
110         10    char *end = NULL;
111         10    char *buf = (char *) calloc(BUFFER_FOR_DATE_SIZE, sizeof(char));
112
113 ✓✓✓✗   30    while (*(date_str + i) != ':' && i != SYMBOLS_FOR_DAYS) {
114         20        buf[i] = date_str[i];
115         20        i++;
116               }
117         10    date_arr[0] = (size_t) strtol(buf, &end, BASE);
118 ✓✗✓✓   10    if (*end != '\0' || date_arr[0] > MAX_DAYS_IN_MONTH) {
119          2        free(buf);
120          2        return -1;
121               }
122
123          8    date_str += (i + 1);
124          8    i = 0;
125
126 ✓✓✓✗   24    while (*(date_str + i) != ':' && i != SYMBOLS_FOR_MONTHS) {
127         16        buf[i] = date_str[i];
128         16        i++;
129               }
130          8    date_arr[1] = (size_t) strtol(buf, &end, BASE);
131 ✓✗✓✓    8    if (*end != '\0' || date_arr[1] > MAX_MONTHS_IN_YEAR) {
132          2        free(buf);
133          2        return -1;
134               }
135
136          6    date_str += (i + 1);
137          6    i = 0;
138
139 ✓✓✓✗   30    while (*(date_str + i) != '\0' && i != SYMBOLS_OF_YEARS) {
140         24        buf[i] = date_str[i];
```

```
141          24         i++;
142                 }
143           6     date_arr[2] = (size_t) strtol(buf, &end, BASE);
144 ✓✗✓✓      6     if (*end != '\0' || date_arr[2] > CURRENT_YEAR) {
145           4         free(buf);
146           4         return -1;
147                 }
148
149           2     free(buf);
150           2     return 0;
151            }
152
153            int will_continue_creating_tasks() {
154                printf("Do you want to create a task(y or n): ");
155
156                char ans[BUFFER_FOR_ANSWER];
157                int res = scanf("%2s", ans);
158                if (res != 1) {
159                    return -1;
160                }
161
162                if (*ans == 'y') {
163                    return 1;
164                }
165
166                return 0;
167            }
168
169            Task *create_task(const size_t *numb_of_task) {
170                if (numb_of_task == NULL) {
171                    return NULL;
172                }
173
174                Task *task = (Task *) calloc(1, sizeof(Task));
175                if (task == NULL) {
176                    return NULL;
177                }
178
179                task->number = *numb_of_task + 1;
180
181                if (read_priority(task) == -1) {
182                    free(task);
183                    return NULL;
184                }
185                if (read_date(task) == -1) {
186                    free(task);
187                    return NULL;
188                }
189                if (read_description(task) == -1) {
190                    free(task);
191                    return NULL;
192                }
193
194                return task;
195            }
196
197            int read_priority(Task *task) {
198                if (task == NULL) {
199                    return -1;
200                }
201                printf("number: %zu\npriority: ", task->number);
202
203                char *end = NULL;
204                char *buf = (char *) calloc(COMMON_BUFFER, sizeof(char));
205                if (buf == NULL) {
206                    return -1;
207                }
208                if (scanf("%10s", buf) != 1) {
209                    free(buf);
210                    return -1;
211                }
212                task->priority = (size_t) strtol(buf, &end, BASE);
213                if (*end != '\0') {
214                    free(buf);
215                    return -1;
216                }
217
```

```
218              free(buf);
219              return 0;
220          }
221
222          int read_date(Task *task) {
223              if (task == NULL) {
224                  return -1;
225              }
226              printf("date(XX:XX:XXXX): ");
227
228              char *date_buf = (char *) calloc(COMMON_BUFFER, sizeof(char));
229              if (date_buf == NULL) {
230                  free(date_buf);
231                  return -1;
232              }
233              if (scanf("%10s", date_buf) != 1) {
234                  free(date_buf);
235                  return -1;
236              }
237              if (parse_date(date_buf, task->date) == -1) {
238                  free(date_buf);
239                  return -1;
240              }
241
242              free(date_buf);
243              return 0;
244          }
245
246          int read_description(Task *task) {
247              if (task == NULL) {
248                  return -1;
249              }
250              printf("description: ");
251
252              task->description = (char *) calloc(SIZE_OF_DESCRIPTION, sizeof(char));
253              if (task->description == NULL) {
254                  return -1;
255              }
256              if (scanf("%99s", task->description) != 1) {
257                  free(task->description);
258                  return -1;
259              }
260
261              return 0;
262          }
263
264          4Tasks *create_array_of_tasks() {
265          4    Tasks *tasks = (Tasks *) calloc(1, sizeof(Tasks));
266   ✗✓     4    if (tasks == NULL) {
267                   return NULL;
268               }
269
270          4    tasks->buffer = (Task *) calloc(START_SIZE_OF_TASKS_BUFFER, sizeof(Task));
271   ✗✓     4    if (tasks->buffer == NULL) {
272                   free(tasks);
273                   return NULL;
274               }
275
276          4    tasks->tasks_amount = 0;
277          4    tasks->cells_amount = 2;
278
279          4    return tasks;
280           }
281
282          2int grow_tasks(Tasks *tasks) {
283   ✗✓     2    if (tasks == NULL) {
284                   return -1;
285               }
286
287          2    tasks->cells_amount *= 2;
288          2    Task *tmp_buffer = (Task *) calloc(tasks->cells_amount, sizeof(Task));
289   ✗✓     2    if (tmp_buffer == NULL) {
290                   return -1;
291               }
292
293          2    tmp_buffer = memcpy(tmp_buffer, tasks->buffer, tasks->tasks_amount * sizeof(Task));
```

```
294    ✗✓    2    if (tmp_buffer == NULL) {
295                     return -1;
296                 }
297
298          2      free(tasks->buffer);
299          2      tasks->buffer = tmp_buffer;
300
301          2      return 0;
302           }
303
304          2  int push_back_task(Tasks *tasks, Task *task) {
305 ✓✗✗✓      2      if (tasks == NULL || task == NULL) {
306                     return -1;
307                 }
308
309    ✗✓    2      if (tasks->cells_amount == tasks->tasks_amount) {
310                     if (grow_tasks(tasks)) {
311                         return -1;
312                     }
313                 }
314
315          2      tasks->buffer[tasks->tasks_amount] = *task;
316          2      free(task);
317          2      tasks->tasks_amount++;
318
319          2      return 0;
320           }
321
322           int buffer_delete(Tasks *tasks) {
323                 if (tasks == NULL) {
324                     return -1;
325                 }
326
327                 for (size_t i = 0; i < tasks->tasks_amount; ++i) {
328                     free(tasks->buffer[i].description);
329                 }
330
331                 free(tasks->buffer);
332                 free(tasks);
333                 return 0;
334           }
```

Generated by: GCOVR (Version 4.2)