

```
1
2 /**
3  * Try these practice string functions to work with pointers.
4  */
5 // Include functions from the standard C library.
6 #include <stdlib.h>
7 // Include string functions (for testing)
8 #include <string.h>
9 // Include functions to support input and output (printing).
10 #include <stdio.h>
11 // Include assertions
12 #include <assert.h>
13
14 /**
15  * Return the length of str, not counting the null terminator
16  * character.
17  *
18  * Precondition: str is a well-formed C string.
19  *
20  * Use array indexing. Do not use any other functions.
21  */
22 int string_length_a(char str[]) {
23     int i = 0;
24     while (str[i]) {
25         i++;
26     }
27     return i;
28 }
29
30 /**
31  * Return the length of str, not counting the null terminator
32  * character.
33  *
34  * Precondition: str is a well-formed C string.
35  *
36  * Do not use array indexing. [] is banned. Use pointer arithmetic instead.
37  * Do not use any other functions.
38  */
39 int string_length_p(char* str) {
40     char* cursor = str;
41     while (*cursor) {
42         cursor++;
43     }
44     return cursor - str;
45 }
46
47 /**
48  * Return 1 if the string given by haystack contains the character
```

```
49 * given by needle.
50 * Return 0 otherwise.
51 *
52 * Precondition: haystack is a valid pointer to a well-formed string.
53 *
54 * Use array indexing.
55 * Do not use any other functions.
56 */
57 int contains_char_a(char* haystack, char needle) {
58     int i = 0;
59     while (haystack[i]) {
60         if (needle == haystack[i]) {
61             return 1;
62         }
63         i++;
64     }
65
66     return 0;
67 }
68
69 /**
70 * Return 1 if the string given by haystack contains the character
71 * given by needle.
72 * Return 0 otherwise.
73 *
74 * Precondition: haystack is a valid pointer to a well-formed string.
75 *
76 * Do not use array indexing. [] is banned. Use pointer arithmetic instead.
77 * Do not use any other functions.
78 */
79 int contains_char_p(char* haystack, char needle) {
80     while (*haystack) {
81         if (needle == *haystack) {
82             return 1;
83         }
84         haystack++;
85     }
86     return 0; // FIXME
87 }
88
89
90 /**
91 * Return 1 if the string given by haystack contains the string given
92 * by needle as a substring.
93 * Return 0 otherwise.
94 *
95 * Precondition: haystack and needle are both valid pointers to
96 * well-formed strings.
```

```
97  *
98  * Do not use any other functions.  There are a few ways to implement
99  * this, some more efficient than others.  Start with a simple
100 * approach.  Optimize if you have time left over at the end of lab.
101 */
102 int contains_string(char* haystack, char* needle) {
103     if (*needle == 0) { //special case if needle is empty string, return tr
104         return 1;
105     }
106     while (*haystack) {
107         if (*haystack == *needle) {
108             char* tempcursor = haystack;
109             char* tempneedle = needle;
110             while (*tempneedle) {
111                 if (*tempcursor != *tempneedle) { //mismatch, exit loop
112                     break;
113                 }
114                 tempcursor++;
115                 tempneedle++;
116             }
117             if (*tempneedle == 0) { //exited loop without mismatch
118                 return 1;
119             }
120         }
121         haystack++;
122     }
123     return 0;
124 }
125
126 /**
127  * Return a pointer to a newly allocated string holding the characters
128  * in haystack starting with the character at index start and ending
129  * just before the character at index end.
130  *
131  * Preconditions:
132  * - haystack is a valid pointer to a well-formed string.
133  * - To begin, assume start and end respect the bounds of haystack:
134  *   - start > 0
135  *   - start < (length of string) - 1
136  *   - end < (length of string) - 1
137  *   - end - start >= 0
138  *
139  * Do not use any other functions besides malloc.
140  */
141 char* substring(char* haystack, int start, int end) {
142     int numbytes = end - start;
143     char* substr = (char*)malloc(numbytes+1);
144     char* startch = haystack+start;
```

```
145     char* tempsubstr = substr;
146     for (int i = 0; i < numbytes;i++) {
147         *tempsubstr++ = *startch++;
148     }
149     *tempsubstr = '\0';
150     return substr;
151 }
152
153 #define N 16
154 static char* test_strings[N] = {
155     "act",
156     "compaction",
157     "actual",
158     "face",
159     "face the action",
160     "face the faction",
161     "factual",
162     "facet",
163     "facetious",
164     "face facet",
165     "face facts facetiously",
166     "effacing",
167     "efface",
168     "aaabb",
169     "aabb",
170     ""
171 };
172
173 void test_string_length(char* fname, int (*fp)(char*)) {
174     for (int i = 0; i < N; i++) {
175         int len = strlen(test_strings[i]);
176         int s = fp(test_strings[i]);
177         if (s != len) {
178             printf("%s(\"%s\") = %d [FAIL] expected %d\n",
179                 fname, test_strings[i], s, len);
180             return;
181         }
182     }
183 }
184
185 void test_contains_char(char* fname, int (*fp)(char*, char)) {
186     for (int i = 0; i < N; i++) {
187         char needle[2];
188         needle[1] = '\0';
189         for (char c = 'a'; c <= 'z'; c++) {
190             needle[0] = c;
191             int contains = strstr(test_strings[i], needle) != NULL;
192             int s = fp(test_strings[i], c);
```

```
193     if (s != contains) {
194         printf("%s(\"%s\", '%c') = %d [FAIL] expected %d\n",
195             fname, test_strings[i], c, s, contains);
196         return;
197     }
198 }
199 }
200 printf("%s [OK]\n", fname);
201 }
202
203 void test_contains_string(char* fname, int (*fp)(char*, char*)) {
204     for (int i = 0; i < N; i++) {
205         for (int j = 0; j < N; j++) {
206             int contains = strstr(test_strings[i], test_strings[j]) != NULL;
207             int s = fp(test_strings[i], test_strings[j]);
208             if (s != contains) {
209                 printf("%s(\"%s\", \"%s\") = %d [FAIL] expected %d\n",
210                     fname, test_strings[i], test_strings[j], s, contains);
211                 return;
212             }
213         }
214     }
215     printf("%s [OK]\n", fname);
216 }
217
218 void test_substring() {
219     char* str = "compaction";
220     int len = strlen(str);
221     for (int start = 0; start < len; start++) {
222         for (int end = start; end <= len; end++) {
223             char* result = substring(str, start, end);
224             if (!result) {
225                 printf("substring(\"%s\", %d, %d) = NULL [FAIL] expected string\n"
226                     str, start, end);
227                 return;
228             }
229             if (result[end - start] != '\0') {
230                 printf("substring(\"%s\", %d, %d) [FAIL] result is missing '\\0' t
231                     str, start, end);
232                 free(result);
233                 return;
234             }
235             for (int i = 0; i < end - start; i++) {
236                 if (result[i] != str[start + i]) {
237                     printf("substring(\"%s\", %d, %d) = %s [FAIL] mismatch at index
238                         str, start, end, result, i);
239                     free(result);
240                     return;

```

```
241     }
242     }
243     free(result);
244 }
245 }
246 printf("substring [OK]\n");
247 }
248
249 /**
250  * Testing driver.
251  */
252 int main(int argc, char** argv) {
253     test_string_length("string_length_a", string_length_a);
254     test_string_length("string_length_p", string_length_p);
255
256     test_contains_char("contains_char_a", contains_char_a);
257     test_contains_char("contains_char_p", contains_char_p);
258
259     test_contains_string("contains_string", contains_string);
260
261     test_substring();
262 }
263
```