```

/**
 * Try these practice string functions to work with pointers.
 */
// Include functions from the standard C library.
#include <stdlib.h>
// Include string functions (for testing)
#include <string.h>
// Include functions to support input and output (printing).
#include <stdio.h>
// Include assertions
#include <assert.h>

/**
 * Return the length of str, not counting the null terminator
 * character.
 *
 * Precondition: str is a well-formed C string.
 *
 * Use array indexing. Do not use any other functions.
 */
int string_length_a(char str[]) {
    int i = 0;
    while (str[i]) {
    i++;
    }
   return i;
}

/**
 * Return the length of str, not counting the null terminator
 * character.
 *
 * Precondition: str is a well-formed C string.
 *
 * Do not use array indexing.  [] is banned.  Use pointer arithmetic instea
 * Do not use any other functions.
 */
int string_length_p(char* str) {
  char* cursor = str;
  while (*cursor) {
      cursor++;
  }
      return cursor - str;
}

/**
 * Return 1 if the string given by haystack contains the character
```

```c
 * given by needle.
 * Return 0 otherwise.
 *
 * Precondition: haystack is a valid pointer to a well-formed string.
 *
 * Use array indexing.
 * Do not use any other functions.
 */
int contains_char_a(char* haystack, char needle) {
    int i = 0;
    while (haystack[i]) {
    if (needle == haystack[i]) {
        return 1;
    }
    i++;
    }

return 0;
}

/**
 * Return 1 if the string given by haystack contains the character
 * given by needle.
 * Return 0 otherwise.
 *
 * Precondition: haystack is a valid pointer to a well-formed string.
 *
 * Do not use array indexing.  [] is banned.  Use pointer arithmetic instea
 * Do not use any other functions.
 */
int contains_char_p(char* haystack, char needle) {
    while (*haystack) {
    if (needle == *haystack) {
        return 1;
    }
    haystack++;
    }
    return 0; // FIXME
}


/**
 * Return 1 if the string given by haystack contains the string given
 * by needle as a substring.
 * Return 0 otherwise.
 *
 * Precondition: haystack and needle are both valid pointers to
 * well-formed strings.
 *
 * Do not use any other functions.  There are a few ways to implement
 * this, some more efficient than others.  Start with a simple
 * approach.  Optimize if you have time left over at the end of lab.
 */
int contains_string(char* haystack, char* needle) {
    if (*needle == 0) { //special case if needle is empty string, return tr
    return 1;
    }
    while (*haystack) {
    if (*haystack == *needle) {
        char* tempcursor = haystack;
        char* tempneedle = needle;
        while (*tempneedle) {
        if (*tempcursor != *tempneedle) { //mismatch, exit loop
            break;
        }
        tempcursor++;
        tempneedle++;
        }
            if (*tempneedle == 0) {  //exited loop without mismatch
          return 1;
            }
    }
    haystack++;
    }
    return 0;
}

/**
 * Return a pointer to a newly allocated string holding the characters
 * in haystack starting with the character at index start and ending
 * just before the character at index end.
 *
 * Preconditions:
 * - haystack is a valid pointer to a well-formed string.
 * - To begin, assume start and end respect the bounds of haystack:
 *    - start > 0
 *    - start < (length of string) - 1
 *    - end < (length of string) - 1
 *    - end - start >= 0
 *
 * Do not use any other functions besides malloc.
 */
char* substring(char* haystack, int start, int end) {
    int numbytes = end - start;
    char* substr = (char*)malloc(numbytes+1);
    char* startch = haystack+start;
```

```c
    char* tempsubstr = substr;
    for (int i = 0; i < numbytes;i++) {
    *tempsubstr++ = *startch++;
    }
    *tempsubstr = '\0';
  return substr;
}

#define N 16
static char* test_strings[N] = {
  "act",
  "compaction",
  "actual",
  "face",
  "face the action",
  "face the faction",
  "factual",
  "facet",
  "facetious",
  "face facet",
  "face facts facetiously",
  "effacing",
  "efface",
  "aaabb",
  "aabb",
  ""
};

void test_string_length(char* fname, int (*fp)(char*)) {
  for (int i = 0; i < N; i++) {
    int len = strlen(test_strings[i]);
    int s = fp(test_strings[i]);
    if (s != len) {
      printf("%s(\"%s\") = %d  [FAIL] expected %d\n",
             fname, test_strings[i], s, len);
      return;
    }
  }
}

void test_contains_char(char* fname, int (*fp)(char*, char)) {
  for (int i = 0; i < N; i++) {
    char needle[2];
    needle[1] = '\0';
    for (char c = 'a'; c <= 'z'; c++) {
      needle[0] = c;
      int contains = strstr(test_strings[i], needle) != NULL;
      int s = fp(test_strings[i], c);
      if (s != contains) {
        printf("%s(\"%s\", '%c') = %d  [FAIL] expected %d\n",
               fname, test_strings[i], c, s, contains);
        return;
      }
    }
  }
  printf("%s  [OK]\n", fname);
}

void test_contains_string(char* fname, int (*fp)(char*, char*)) {
  for (int i = 0; i < N; i++) {
    for (int j = 0; j < N; j++) {
      int contains = strstr(test_strings[i], test_strings[j]) != NULL;
      int s = fp(test_strings[i], test_strings[j]);
      if (s != contains) {
        printf("%s(\"%s\", \"%s\") = %d  [FAIL] expected %d\n",
               fname, test_strings[i], test_strings[j], s, contains);
        return;
      }
    }
  }
  printf("%s  [OK]\n", fname);
}

void test_substring() {
  char* str = "compaction";
  int len = strlen(str);
  for (int start = 0; start < len; start++) {
    for (int end = start; end <= len; end++) {
      char* result = substring(str, start, end);
      if (!result) {
        printf("substring(\"%s\", %d, %d) = NULL  [FAIL] expected string\n"
               str, start, end);
        return;
      }
      if (result[end - start] != '\0') {
        printf("substring(\"%s\", %d, %d)  [FAIL] result is missing '\\0' t
               str, start, end);
        free(result);
        return;
      }
      for (int i = 0; i < end - start; i++) {
        if (result[i] != str[start + i]) {
          printf("substring(\"%s\", %d, %d) = %s  [FAIL] mismatch at index 
                 str, start, end, result, i);
          free(result);
          return;
        }
      }
      free(result);
    }
  }
  printf("substring  [OK]\n");
}

/**
 * Testing driver.
 */
int main(int argc, char** argv) {
  test_string_length("string_length_a", string_length_a);
  test_string_length("string_length_p", string_length_p);

  test_contains_char("contains_char_a", contains_char_a);
  test_contains_char("contains_char_p", contains_char_p);

  test_contains_string("contains_string", contains_string);

  test_substring();
}

```