**Name – Hla Mynit Myat**

**ID - 185923216**

**A list of the tests**

1. Test for startsWith function

char s1[] = { "upended" };

char prefix[] = { "up" };

. check if s1 starts with prefix

. should return true(1)

.result should be “upended does start with up.”

. the startsWith function correctly determined that the string "upended" starts with the prefix "up". Therefore, the function is working as expected.

. No bug identified

1. Test for endsWith function

char s1[] = { "upended" };

char suffix[] = { "ed" };

.check if s1 ends with suffix

. should return true (1)

. result should be "upended does end with ed."

. The endsWith function correctly determined that the string "upended" ends with the suffix "ed". Therefore, the function is working as expected.

. No bug identified

**After inspecting the code**

Additional bugs

**Bug 1.Buffer overflow**

The code does not handle the case when the length of the prefix or suffix is greater than the size of the buffer buf in the startsWith function. This could lead to a buffer overflow and undefined behavior.  
  
char buf[20];

int sz = strlen(prefix);

for (i = 0; i < sz; i++){

buf[i] = s[i];

}

The test will likely result in a program crash or undefined behavior due to buffer overflow.

In the startsWith function, if the length of the prefix is larger than the buffer size (20), the for loop will attempt to write beyond the bounds of the buf array. This can overwrite adjacent memory, leading to program crashes or unpredictable behavior.

To fix the buffer overflow bug,

We need to ensure that the size of buf is sufficient to hold the prefix.

int startsWith(const char s[], const char prefix[]) {

int sz = strlen(prefix);

return 0 == strncmp(s, prefix, sz);

}

We replace the strcmp function with strncmp , which compares only the first sz characters of s and prefix .

**Bug 2.Do not Handle Empty Strings**

The startsWith and endsWith functions do not handle empty strings.

int startsWith(const char s[], const char prefix[])

{

// ... //

}

int endsWith(const char s[], const char suffix[])

{

// ... //

}

The original code did not handle the case when either the prefix or suffix is an empty string. This could lead to unexpected behavior or incorrect results.

To fix this bug ,  
  
We have to check if the prefix or suffix is empty and returning true in those cases. This ensures that empty strings are correctly handled and avoids any erroneous results.

int startsWith(const char s[], const char prefix[])

{

if (strlen(prefix) == 0) {

return 1; // Return true if the prefix is empty

}

// ..//

}

int endsWith(const char s[], const char suffix[])

{

if (strlen(suffix) == 0) {

return 1; // Return true if the suffix is empty

}

// ..//

}

**Bug 3.Does not handle when the prefix is longer than the input string**

The startsWith function does not handle cases where the prefix is longer than the input string.

int startsWith(const char s[], const char prefix[])

{

// ... //

}

The original startsWith function did not handle cases where the prefix is longer than the input string s . This could lead to incorrect results as the function would compare more characters than available in s .

To fix this Bug,

We have to check if the size of the prefix exceeds the size of s and returning false in those cases. This ensures that the function works correctly even when the prefix is longer than the input string.

int startsWith(const char s[], const char prefix[])

{

int sz = strlen(prefix);

if (sz > strlen(s)) {

return 0; // Return false if the prefix is longer than the input string

}

return 0 == strncmp(s, prefix, sz);

}

**Bug 4. Incorrect return values in functions**

int startsWith(const char s[], const char prefix[])

{

// ... //

}

int endsWith(const char s[], const char suffix[])

{

// ... //

}

The original code used 0 == strcmp(...) and 0 == strncmp(...) to compare the strings and return 1 or 0 based on the comparison result. However, strcmp and strncmp return 0 when the strings are equal, which is the opposite of the expected return values (1 for true and 0 for false).

To fix this Bug,

We have to change the return statements to explicitly check if the comparison result is 0 and return 1 or 0 accordingly.

int startsWith(const char s[], const char prefix[])

{

int sz = strlen(prefix);

if (sz > strlen(s)) {

return 0; // Return false if the prefix is longer than the input string

}

return strncmp(s, prefix, sz) == 0 ? 1 : 0; // Return 1 if the strings are equal, otherwise return 0

}

int endsWith(const char s[], const char suffix[])

{

int sz = strlen(suffix);

int slen = strlen(s);

if (sz > slen) {

return 0; // Return false if the suffix is longer than the input string

}

return strcmp(s + slen - sz, suffix) == 0 ? 1 : 0; // Return 1 if the strings are equal, otherwise return 0;

}

Now, the startsWith and endsWith functions will return the correct values (1 for true and 0 for false) based on the comparison results.

**A Reflection, Research and Assessment**

Bug 1 - By running the code and analyzing the behavior, it became clear that the buffer overflow occurred when the prefix was longer than the buffer size. Inspection might not have directly revealed this bug unless a thorough examination of the code, specifically the buffer size and usage, was conducted.

Reflection for bug 1- In this case, testing was more effective in identifying the buffer overflow bug. The dynamic execution of the code during testing exposed the issue. To improve the inspection technique, a more careful review of buffer sizes and potential overflow scenarios could be conducted.

Bug 2 - By running the code with empty strings as inputs, it became evident that the functions did not handle these cases correctly. Inspection could have also identified this bug by closely examining the code logic and considering edge cases like empty strings.

Reflection for bug 2 - Both testing and inspection techniques were effective in identifying this bug. However, inspection might have been more efficient in this case, as it is relatively straightforward to identify missing checks for empty strings during code review. To improve the testing technique, more comprehensive test cases covering different scenarios, including empty strings, could be included.

Bug 3 - By running the code with a prefix longer than the input string, it became clear that the function did not handle this case correctly. Inspection could have also identified this bug by carefully reviewing the code logic and considering scenarios where the prefix is longer than the input string.

Reflection for bug 3 – In this case, analyzing the code logic and considering potential edge cases during review could reveal the issue. To improve the testing technique, additional test cases specifically targeting scenarios with longer prefixes could be included.

Bug 4 - By running the code and comparing the actual results with the expected results, it became evident that the return values were incorrect. Inspection could have also identified this bug by carefully reviewing the code and scrutinizing the return statements.

Reflection for bug 4 – In this case, reviewing the code and identifying incorrect return values could be done during inspection. To improve the testing technique, more comprehensive test cases covering different scenarios could be included to ensure accurate return values.

The bugs in this assignment were not particularly difficult to find.

1. Understanding of expected behavior
2. Reviewing code logic
3. Testing with different inputs
4. Comparing actual results with expected results

By employing these techniques and approaches, the bugs were found relatively quickly.