Test name	
Test Description	
Line(s)	
Code	
Test Results	

Explanations

Bug or not?

test 1 - 1

Let's find out how the output comes out if there are words in s1 that are blank in suffix

36 and 37

```
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include <string.h>

/*** Determine if the string s starts with the string prefix.
* @param s - the string to see if it starts with prefix
* @param prefix - the prefix to test
* @returns true if the string begins with the prefix.
*/
int startsWith(const char s[], const char prefix[]) {
    char buf[20];
    int i;
    int sz = strlen(prefix);

    for (i = 0; i < sz; i++) {
        buf[i] = s[i];
    }
    buf[sz] = '\0';
    printf("%d\n", strcmp(buf, prefix));
    return 0 == strcmp(buf, prefix);
}

/*** Determine if the string s ends with the string suffix.
* @param s - the string to see if it ends with suffix
* @param suffix - the suffix to test
* @returns true if the string ends with the suffix.
*/
int endsWith(const char s[], const char suffix[]) {
    int sz = strlen(suffix);
    int slen = strlen(s);

    return 0 == strcmp(s + slen - sz, suffix);
}

int main(void) {
    char s1[] = { "upended" };
    char prefix[] = { "" };
```

```
upended does  start with
upended does  end with
```

First of all, the s1 variable contains a string, but the prefix variable does not have a string assigned to it. When you enter the startsWith function, the s parameter represents the s1 string, and the prefix parameter represents the uninitialized prefix variable. Then, a buf variable of string type and an i variable are declared. The sz variable is initialized with the length of the prefix string. However, since sz is 0, the loop does not execute. Finally, a null character is assigned to the 0th index of buf, and since both buf and prefix are empty, the comparison 0 == strcmp(buf, prefix) evaluates to true (0), which leads to "not" not being printed.

In summary, since the prefix variable is not assigned a string and its length is 0, the loop in the startsWith function does not execute, resulting in a successful comparison between buf and prefix, and hence the output "not" is not printed.

Bug

SFT 221

Workshop 1

Name:	Gyeongrok oh
ID:	119140226
Email:	goh3@myseneca.ca
Section:	NBB

Authenticity Declaration:
I declare this submission is the result of my own work and has not been shared with any other student or 3rd party content provider. This submitted piece of work is entirely of my own creation.

Test name	
Test Description	
Line(s)	
Code	
Test Results	

Explanations
Bug or not?

test 1 - 1

Let's find out how the output comes out if there are words in s1 that are blank in suffix

36 and 37

```
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include <string.h>

/*** Determine if the string s starts with the string prefix.
* @param s - the string to see if it starts with prefix
* @param prefix - the prefix to test
* @returns true if the string begins with the prefix.
*/
int startsWith(const char s[], const char prefix[]) {
   char buf[20];
   int i;
   int sz = strlen(prefix);

   for (i = 0; i < sz; i++) {
      buf[i] = s[i];
   }
   buf[sz] = '\0';
   printf("%d\n", strcmp(buf, prefix));
   return 0 == strcmp(buf, prefix);
}

/*** Determine if the string s ends with the string suffix.
* @param s - the string to see if it ends with suffix
* @param suffix - the suffix to test
* @returns true if the string ends with the suffix.
*/
int endsWith(const char s[], const char suffix[]) {
   int sz = strlen(suffix);
   int slen = strlen(s);

   return 0 == strcmp(s + slen - sz, suffix);
}

int main(void) {
   char s1[] = { "upended" };
   char prefix[] = { "" };
```

```
upended does  start with
upended does  end with
```

First of all, the s1 variable contains a string, but the prefix variable does not have a string assigned to it. When you enter the startsWith function, the s parameter represents the s1 string, and the prefix parameter represents the uninitialized prefix variable. Then, a buf variable of string type and an i variable are declared. The sz variable is initialized with the length of the prefix string. However, since sz is 0, the loop does not execute. Finally, a null character is assigned to the 0th index of buf, and since both buf and prefix are empty, the comparison 0 == strcmp(buf, prefix) evaluates to true (0), which leads to "not" not being printed.

In summary, since the prefix variable is not assigned a string and its length is 0, the loop in the startsWith function does not execute, resulting in a successful comparison between buf and prefix, and hence the output "not" is not printed.

Bug

Test name	
Test Description	
Line(s)	
Code	
Test Results	

Explanations
Bug or not?

Test 1 - 2

Let's analyze the code when the number of characters and prefixes in s1 exceeds 20 The result of the

36 and 37

```
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include <string.h>

/*** Determine if the string s starts with the string prefix.
* @param s - the string to see if it starts with prefix
* @param prefix - the prefix to test
* @returns true if the string begins with the prefix.
*/
int startsWith(const char s[], const char prefix[]) {
   char buf[20];
   int i;
   int sz = strlen(prefix);
   if (strlen(s) == strlen(prefix)) {
      for (i = 0; i < sz; i++) {
         buf[i] = s[i];
      }
      buf[sz] = '\0';

      return 0 == strcmp(buf, prefix);
   }
   return 0;
}


/*** Determine if the string s ends with the string suffix.
* @param s - the string to see if it ends with suffix
* @param suffix - the suffix to test
* @returns true if the string ends with the suffix.
*/
int endsWith(const char s[], const char suffix[]) {
   int sz = strlen(suffix);
   int slen = strlen(s);

   return 0 == strcmp(s + slen - sz, suffix);
}
```

error

char s1[] = { "upended" };: Declares an array s1 and initializes it with the string literal "upended". The string "upended" is stored in the array s1.

char prefix[] = { "upendedupendedupended" };: Declares an array prefix and initializes it with the string literal "upendedupendedupended". The string "upendedupendedupended" is stored in the array prefix.

char suffix[] = { "upendedupendedupended" };: Declares an array suffix and initializes it with the string literal "upendedupendedupended". The string "upendedupendedupended" is stored in the array suffix.

The printf statement is used to print the result of checking whether s1 starts with prefix. It uses the following format string: "%s does %s start with %s\n".

The first %s is a placeholder that will be replaced with the content of s1, which is the string "upended".
The second %s is a placeholder that will be replaced with either an empty string ("") if startsWith(s1, prefix) returns true, indicating that s1 starts with prefix, or with the string "not" if the condition is false.

bug

print should not be included 'not'

Test name	
Test Description	
Line(s)	
Code	
Test Results	

Explanations
Bug or not?

Test 1 -3

If the two characters in the s1 string are the same as the prefix, but there is a null character value in the middle,

36 and 37

```
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include <string.h>

/*** Determine if the string s starts with the string prefix.
* @param s - the string to see if it starts with prefix
* @param prefix - the prefix to test
* @returns true if the string begins with the prefix.
*/
int startsWith(const char s[], const char prefix[]) {
    char buf[20];
    int i;
    int sz = strlen(prefix);

    for (i = 0; i < sz; i++) {
        buf[i] = s[i];
    }
    buf[sz] = '\0';
    printf("%d\n", strcmp(buf, prefix));
    return 0 == strcmp(buf, prefix);
}



/*** Determine if the string s ends with the string suffix.
* @param s - the string to see if it ends with suffix
* @param suffix - the suffix to test
* @returns true if the string ends with the suffix.
*/
int endsWith(const char s[], const char suffix[]) {
    int sz = strlen(suffix);
    int slen = strlen(s);

    return 0 == strcmp(s + slen - sz, suffix);
}

int main(void) {
```

```
upended does  start with up
upended does not end with upendedupendedupended
```

In the startsWith function, the s1 and prefix are passed as parameters. The compiler starts reading from left to right until it encounters a null character, which signifies the end of a line. In this case, the null character is encountered after the second character, so the sz variable is assigned a value of 2.

Next, the for loop iterates twice, copying each character from s1 to the buf array. As a result, "up" is stored in buf. The null character ('\0') is then added at the third index of buf, terminating it as a null-terminated string.

When comparing buf and prefix using the strcmp function, they are found to be different. Therefore, the condition 0 == strcmp(buf, prefix) evaluates to false, and the corresponding "not" statement is printed.

bug

our code should have a 'not'

Line #

Original code

Fixed code	
Test #	
Explanations	

41

```
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include <string.h>

/*** Determine if the string s starts with the string prefix.
* @param s - the string to see if it starts with prefix
* @param prefix - the prefix to test
* @returns true if the string begins with the prefix.
*/
int startsWith(const char s[], const char prefix[]) {
   char buf[20];
   int i;
   int sz = strlen(prefix);

   for (i = 0; i < sz; i++) {
      buf[i] = s[i];
   }
   buf[sz] = '\0';
   printf("%d\n", strcmp(buf, prefix));
   return 0 == strcmp(buf, prefix);
}

/*** Determine if the string s ends with the string suffix.
* @param s - the string to see if it ends with suffix
* @param suffix - the suffix to test
* @returns true if the string ends with the suffix.
*/
int endsWith(const char s[], const char suffix[]) {
   int sz = strlen(suffix);
   int slen = strlen(s);

   return 0 == strcmp(s + slen - sz, suffix);
}

int main(void) {
   char s1[] = { "upended" };
   char prefix[] = { "" };
```

```
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include <string.h>

/*** Determine if the string s starts with the string prefix.
* @param s - the string to see if it starts with prefix
* @param prefix - the prefix to test
* @returns true if the string begins with the prefix.
*/
int startsWith(const char s[], const char prefix[]) {
    char buf[20];
    int i;
    int sz = strlen(prefix);
    if (strlen(s) == strlen(prefix)) {
        for (i = 0; i < sz; i++) {
            buf[i] = s[i];
        }
        buf[sz] = '\0';
        printf("%d", strcmp(buf, prefix));
        return 0 == strcmp(buf, prefix);
    }
    return 0;
}


/*** Determine if the string s ends with the string suffix.
* @param s - the string to see if it ends with suffix
* @param suffix - the suffix to test
* @returns true if the string ends with the suffix.
*/
int endsWith(const char s[], const char suffix[]) {
    int sz = strlen(suffix);
    int slen = strlen(s);

    return 0 == strcmp(s + slen - sz, suffix);
}
```

Test 1

The provided code includes the startsWith function, which determines whether the string s starts with the string

However, there is a bug in the code when prefix is an empty string. In such cases, both strlen(s) and strlen(prefix

The endsWith function determines whether the string s ends with the string suffix. Inside the function, it calcula

; prefix. Inside the function, a character buffer named buf with a size of 20 bytes is created. The function cop

) will be 0. Therefore, the condition if (strlen(s) != strlen(prefix)) becomes false, and the function erroneousl

tes the length sz of the suffix string using the strlen() function. Then it compares the substring of s starting fr ɔies the first characters of s corresponding to the length of prefix into buf. It then compares buf and prefix us

y returns 1. To fix this issue, the comparison operator in the if statement should be changed to ==. It should

rom the index obtained by subtracting sz from the length of s with the suffix string using strcmp() function. Tl ing the strcmp() function to check if they are equal and returns the result.

be written as if (strlen(s) == strlen(prefix)).

he function returns the result of this comparison.

Line #
Original code

Fixed code	
Test #	
Explanations	

41

```
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include <string.h>

/*** Determine if the string s starts with the string prefix.
* @param s - the string to see if it starts with prefix
* @param prefix - the prefix to test
* @returns true if the string begins with the prefix.
*/
int startsWith(const char s[], const char prefix[]) {
   char buf[20];
   int i;
   int sz = strlen(prefix);

   for (i = 0; i < sz; i++) {
      buf[i] = s[i];
   }
   buf[sz] = '\0';
   printf("%d\n", strcmp(buf, prefix));
   return 0 == strcmp(buf, prefix);
}

/*** Determine if the string s ends with the string suffix.
* @param s - the string to see if it ends with suffix
* @param suffix - the suffix to test
* @returns true if the string ends with the suffix.
*/
int endsWith(const char s[], const char suffix[]) {
   int sz = strlen(suffix);
   int slen = strlen(s);

   return 0 == strcmp(s + slen - sz, suffix);
}

int main(void) {
   char s1[] = { "upended" };
   char prefix[] = { "" };
```

```
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include <string.h>

/*** Determine if the string s starts with the string prefix.
* @param s - the string to see if it starts with prefix
* @param prefix - the prefix to test
* @returns true if the string begins with the prefix.
*/
int startsWith(const char s[], const char prefix[]) {
    char buf[20];
    int i;
    int sz = strlen(prefix);
    if (strlen(s) == strlen(prefix)) {
        for (i = 0; i < sz; i++) {
            buf[i] = s[i];
        }
        buf[sz] = '\0';
        printf("%d", strcmp(buf, prefix));
        return 0 == strcmp(buf, prefix);
    }
    return 0;
}


/*** Determine if the string s ends with the string suffix.
* @param s - the string to see if it ends with suffix
* @param suffix - the suffix to test
* @returns true if the string ends with the suffix.
*/
int endsWith(const char s[], const char suffix[]) {
    int sz = strlen(suffix);
    int slen = strlen(s);

    return 0 == strcmp(s + slen - sz, suffix);
}
```

Test 1

The provided code includes the startsWith function, which determines whether the string s starts with the string

However, there is a bug in the code when prefix is an empty string. In such cases, both strlen(s) and strlen(prefix

The endsWith function determines whether the string s ends with the string suffix. Inside the function, it calcula

; prefix. Inside the function, a character buffer named buf with a size of 20 bytes is created. The function cop

:) will be 0. Therefore, the condition if (strlen(s) != strlen(prefix)) becomes false, and the function erroneousl

tes the length sz of the suffix string using the strlen() function. Then it compares the substring of s starting fr ɔies the first characters of s corresponding to the length of prefix into buf. It then compares buf and prefix us

y returns 1. To fix this issue, the comparison operator in the if statement should be changed to ==. It should

ːom the index obtained by subtracting sz from the length of s with the suffix string using strcmp() function. Tl

;ing the strcmp() function to check if they are equal and returns the result.

be written as if (strlen(s) == strlen(prefix)).

he function returns the result of this comparison.

Line #
Original code

Fixed code	
Test #	
Explanations	

18

```
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include <string.h>

/*** Determine if the string s starts with the string prefix.
* @param s - the string to see if it starts with prefix
* @param prefix - the prefix to test
* @returns true if the string begins with the prefix.
*/
int startsWith(const char s[], const char prefix[]) {
   char buf[20];
   int i;
   int sz = strlen(prefix);

   for (i = 0; i < sz; i++) {
      buf[i] = s[i];
   }
   buf[sz] = '\0';

   return 0 == strcmp(buf, prefix);
}


/*** Determine if the string s ends with the string suffix.
* @param s - the string to see if it ends with suffix
* @param suffix - the suffix to test
* @returns true if the string ends with the suffix.
*/
int endsWith(const char s[], const char suffix[]) {
   int sz = strlen(suffix);
   int slen = strlen(s);

   return 0 == strcmp(s + slen - sz, suffix);
}

int main(void) {
   char s1[] = { "upended" };
```

```
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include <string.h>

/*** Determine if the string s starts with the string prefix.
* @param s - the string to see if it starts with prefix
* @param prefix - the prefix to test
* @returns true if the string begins with the prefix.
*/
int startsWith(const char s[], const char prefix[]) {
   int sz = strlen(prefix);
   return strncmp(s, prefix, sz) == 0;
}



/*** Determine if the string s ends with the string suffix.
* @param s - the string to see if it ends with suffix
* @param suffix - the suffix to test
* @returns true if the string ends with the suffix.
*/
int endsWith(const char s[], const char suffix[]) {
   int sz = strlen(suffix);
   int slen = strlen(s);

   return 0 == strcmp(s + slen - sz, suffix);
}

int main(void) {
   char s1[] = { "upended" };
   char prefix[] = { "upendedupendedupended" };
   char suffix[] = { "upendedupendedupended" };

   printf("%s does %s start with %s\n", s1, startsWith(s1, prefix) ? "" : "not", prefix);
   printf("%s does %s end with %s\n", s1, endsWith(s1, suffix) ? "" : "not", suffix);

   return 0;
```

Test 1 - 2

The buffer error in the code occurs in the startsWith function. The main reason for the buffer error is that the size of the buf array is smaller than the length of the prefix string.

To fix the buffer error, one approach is to use the strncmp function. The strncmp function compares the first n characters of two strings. We can utilize it to modify the startsWith function as follows:

By using strncmp, we compare the sz characters of s and prefix. If the result is 0, it means that the s string starts with the prefix string.

This modification ensures that the buffer size is not exceeded and eliminates the buffer error.

Line #	
Original code	
Fixed code	
Test #	
Explanations	

41

```
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include <string.h>

/*** Determine if the string s starts with the string prefix.
* @param s - the string to see if it starts with prefix
* @param prefix - the prefix to test
* @returns true if the string begins with the prefix.
*/
int startsWith(const char s[], const char prefix[]) {
    char buf[20];
    int i;
    int sz = strlen(prefix);

    for (i = 0; i < sz; i++) {
        buf[i] = s[i];
    }
    buf[sz] = '\0';

    return 0 == strcmp(buf, prefix);
}



/*** Determine if the string s ends with the string suffix.
* @param s - the string to see if it ends with suffix
* @param suffix - the suffix to test
* @returns true if the string ends with the suffix.
*/
int endsWith(const char s[], const char suffix[]) {
    int sz = strlen(suffix);
    int slen = strlen(s);

    return 0 == strcmp(s + slen - sz, suffix);
}

int main(void) {
```

In such a case, the code cannot be fixed because it is intentionally forced to cut the code

Test 1 - 3

When we define string literals in C (e.g., "up", "end", etc.), the compiler automatically adds a null character (\0) a
In fact, null characters are used as string terminators in C, indicating the end of a string.

at the end of the string to properly terminate it.

testing and inspection both have their merits in identifying bugs. Testing is more effective at uncovering runtime errors and dynamic issues, while inspection is valuable for analyzing code structure and logic. Combining the two approaches and incorporating peer reviews can enhance the overall bug identification process.

Did you find it difficult to find the bugs in this assignment? If not, what helped find them quickly? If you did find it difficult, what made finding the bugs so difficult?

Having well-defined requirements and specifications can help guide the testing process and provide a clear benchmark for identifying deviations or errors.
Applying a systematic and comprehensive testing approach, including boundary testing, input validation, and edge case scenarios, can increase the likelihood of finding bugs.
Utilizing debugging tools, IDEs, and techniques such as stepping through the code, setting breakpoints, and inspecting variables can help pinpoint the location and cause of bugs more efficiently.
Engaging in code reviews, seeking input from peers or more experienced developers, and leveraging collective knowledge and perspectives can significantly aid in finding bugs. In conclusion, the difficulty of finding bugs in code assignments can vary based on several factors. Clear specifications, systematic testing, proper documentation, and collaborative efforts can contribute to a more efficient bug-finding process.