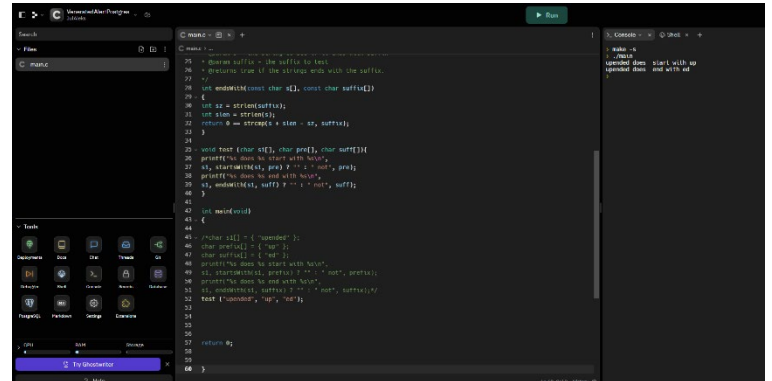


[git@github.com:JulAleks/Sft.git](https://github.com:JulAleks/Sft.git)

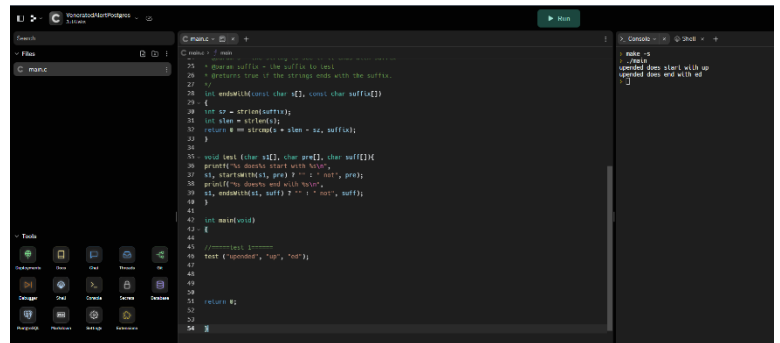
## PART 1 – TEST CASES

- My initial approach to testing was to create a function that would use char variables for the test case rather than duplicating the same code repeatedly. The initial test was to test that the testing function works and generates the same output as the initial code, to make sure that the testing function won't introduce new possible bugs.
- After confirming that my test function is working properly, I have moved to examine the output. I have found that there are two extra spaces between “does” and “start”, and “does” and “end”. Although it might not be an “error”, it does stick out from the format point of view.



```
1 // does_start_end.c
2 #include <stdio.h>
3 #include <string.h>
4
5 // does_start_end: checks if the string starts with 'does' and ends with 'end'
6 // returns true if the string starts with 'does' and ends with 'end'
7 // returns false otherwise
8
9 bool does_start_end(char s[]) {
10     if (s[0] != 'd') return false;
11     if (s[strlen(s)-1] != 'd') return false;
12     return true;
13 }
14
15 void test(char s[], char p[], char r[]) {
16     printf("does %s start with %s", s, p);
17     if (does_start_end(s, p) == r) printf("true");
18     printf("does %s end with %s", s, p);
19     if (does_start_end(s, p) == r) printf("true");
20 }
21
22 int main(void) {
23     test("does start with up", "does", "true");
24     test("end with up", "end", "true");
25 }
26
27 return 0;
28 }
```

```
1 // does_start_end.c
2 #include <stdio.h>
3 #include <string.h>
4
5 // does_start_end: checks if the string starts with 'does' and ends with 'end'
6 // returns true if the string starts with 'does' and ends with 'end'
7 // returns false otherwise
8
9 bool does_start_end(char s[]) {
10     if (s[0] != 'd') return false;
11     if (s[strlen(s)-1] != 'd') return false;
12     return true;
13 }
14
15 void test(char s[], char p[], char r[]) {
16     printf("does %s start with %s", s, p);
17     if (does_start_end(s, p) == r) printf("true");
18     printf("does %s end with %s", s, p);
19     if (does_start_end(s, p) == r) printf("true");
20 }
21
22 int main(void) {
23     test("does start with up", "does", "true");
24     test("end with up", "end", "true");
25 }
26
27 return 0;
28 }
```

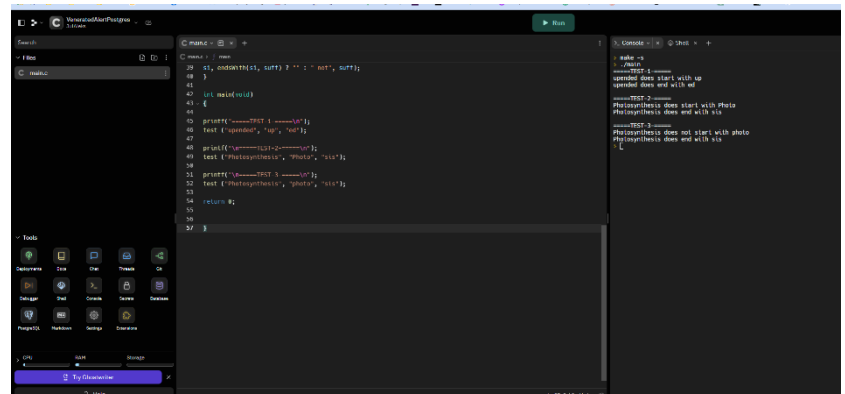


```
1 // does_start_end.c
2 #include <stdio.h>
3 #include <string.h>
4
5 // does_start_end: checks if the string starts with 'does' and ends with 'end'
6 // returns true if the string starts with 'does' and ends with 'end'
7 // returns false otherwise
8
9 bool does_start_end(char s[]) {
10     if (s[0] != 'd') return false;
11     if (s[strlen(s)-1] != 'd') return false;
12     return true;
13 }
14
15 void test(char s[], char p[], char r[]) {
16     printf("does %s start with %s", s, p);
17     if (does_start_end(s, p) == r) printf("true");
18     printf("does %s end with %s", s, p);
19     if (does_start_end(s, p) == r) printf("true");
20 }
21
22 int main(void) {
23     test("does start with up", "does", "true");
24     test("end with up", "end", "true");
25 }
26
27 return 0;
28 }
```

```
1 // does_start_end.c
2 #include <stdio.h>
3 #include <string.h>
4
5 // does_start_end: checks if the string starts with 'does' and ends with 'end'
6 // returns true if the string starts with 'does' and ends with 'end'
7 // returns false otherwise
8
9 bool does_start_end(char s[]) {
10     if (s[0] != 'd') return false;
11     if (s[strlen(s)-1] != 'd') return false;
12     return true;
13 }
14
15 void test(char s[], char p[], char r[]) {
16     printf("does %s start with %s", s, p);
17     if (does_start_end(s, p) == r) printf("true");
18     printf("does %s end with %s", s, p);
19     if (does_start_end(s, p) == r) printf("true");
20 }
21
22 int main(void) {
23     test("does start with up", "does", "true");
24     test("end with up", "end", "true");
25 }
26
27 return 0;
28 }
```

To fix that issue, there would be the space between “does” and “%s” in both statements. Since if the string begins with the test case, there is no need to add anything to the word “does”, and “” will be used in that case. In addition, if the tested string does not represent the beginning or the end, there is already an added space in the “not” string.

- Test 2 and 3 were examining the function using other values, in which the first letter is capitalized. The examined string: “Photosynthesis”, “Photo”, “sis”, vs another version of using “photo” as lower cases. Since C is case sensitive language therefore in test 2 the output stated that the string begins with “Photo” vs test 3 in which it stated is does not begin with “photo”.

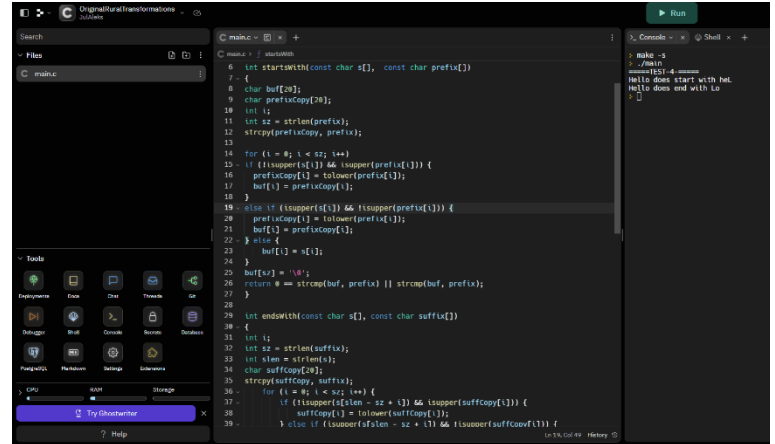


```
1 // does_start_end.c
2 #include <stdio.h>
3 #include <string.h>
4
5 // does_start_end: checks if the string starts with 'does' and ends with 'end'
6 // returns true if the string starts with 'does' and ends with 'end'
7 // returns false otherwise
8
9 bool does_start_end(char s[]) {
10     if (s[0] != 'd') return false;
11     if (s[strlen(s)-1] != 'd') return false;
12     return true;
13 }
14
15 void test(char s[], char p[], char r[]) {
16     printf("does %s start with %s", s, p);
17     if (does_start_end(s, p) == r) printf("true");
18     printf("does %s end with %s", s, p);
19     if (does_start_end(s, p) == r) printf("true");
20 }
21
22 int main(void) {
23     test("does start with up", "does", "true");
24     test("end with up", "end", "true");
25 }
26
27 return 0;
28 }
```

```
1 // does_start_end.c
2 #include <stdio.h>
3 #include <string.h>
4
5 // does_start_end: checks if the string starts with 'does' and ends with 'end'
6 // returns true if the string starts with 'does' and ends with 'end'
7 // returns false otherwise
8
9 bool does_start_end(char s[]) {
10     if (s[0] != 'd') return false;
11     if (s[strlen(s)-1] != 'd') return false;
12     return true;
13 }
14
15 void test(char s[], char p[], char r[]) {
16     printf("does %s start with %s", s, p);
17     if (does_start_end(s, p) == r) printf("true");
18     printf("does %s end with %s", s, p);
19     if (does_start_end(s, p) == r) printf("true");
20 }
21
22 int main(void) {
23     test("does start with up", "does", "true");
24     test("end with up", "end", "true");
25 }
26
27 return 0;
28 }
```

**Course:** SFT221 | **Workshop:** 1 | **Professor:** Robin Huang  
**Section:** NEE | **Student Name:** Julia Alekseev | **Student ID:** 051292134

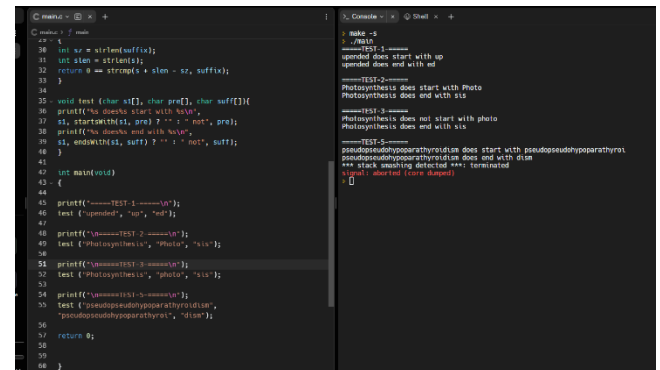
- There is a potential for a user error. For a user Hello or hello, can be the same as one user may be less sensitive to case sensitivity. Depending on if we would like the code to be case sensitive, we can leave it as it is. If we would like the code not to be case sensitive, we can modify the `int startsWith(const char s[], const char prefix[])` and `int endsWith(const char s[], const char suffix[])` function to treat lower and upper cases as the same, this way it will effect both prefix and suffix.



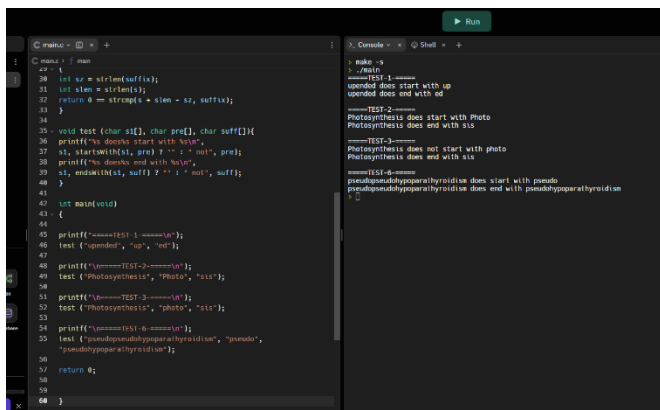
```
1 int startsWith(const char s[], const char prefix[])
2 {
3     char buf[20];
4     char prefixCopy[20];
5     int i;
6     int sz = strlen(prefix);
7     strcpy(prefixCopy, prefix);
8     for (i = 0; i < 52; i++)
9     {
10         if (iisupper(s[i]) && iisupper(prefix[i])) {
11             prefixCopy[i] = tolower(prefix[i]);
12             buf[i] = prefixCopy[i];
13         }
14     }
15     if (iisupper(s[i]) && iisupper(prefix[i])) {
16         prefixCopy[i] = tolower(prefix[i]);
17         buf[i] = prefixCopy[i];
18     }
19     buf[i] = s[i];
20     return 0 == strcmp(buf, prefix) || strcmp(buf, prefix);
21 }
22
23 int endsWith(const char s[], const char suffix[])
24 {
25     int i;
26     int sz = strlen(suffix);
27     int slen = strlen(s);
28     char suffixCopy[sz];
29     strcpy(suffixCopy, suffix);
30     for (i = 0; i < 52; i++) {
31         if (iisupper(s[slen - sz + i]) && iisupper(suffixCopy[i])) {
32             suffixCopy[i] = tolower(suffixCopy[i]);
33             if (iisupper(s[slen - sz + i]) && iisupper(suffixCopy[i])) {
34                 suffixCopy[i] = tolower(suffixCopy[i]);
35             }
36         }
37     }
38     return 0 == strcmp(s + slen - sz, suffixCopy) || strcmp(s + slen - sz, suffixCopy);
39 }
```

As there is no indication that this modification is needed, the following tests would be done using the original function, and the c code would be provided separately for the above test as “test4.c”.

- Tests 5 and 6 are looking into longer words. Although we as programmers can see the code testing as white box, we are aware that the memory allocated for buff is 19 characters plus null terminator, 20, that represents the prefix. A black box tester or the user might not be aware of those limitations and therefore can stumble upon run time memory error. As can see in the photo attached. It is important to note that this does not affect the testing of the suffix as it does not have visible limitations. Both testing can be seen in the photos



```
1 // test4.c
2 #include <stdio.h>
3 #include <string.h>
4 #include <ctype.h>
5
6 int startsWith(const char s[], const char prefix[])
7 {
8     char buf[20];
9     char prefixCopy[20];
10    int i;
11    int sz = strlen(prefix);
12    strcpy(prefixCopy, prefix);
13    for (i = 0; i < 52; i++)
14    {
15        if (iisupper(s[i]) && iisupper(prefix[i])) {
16            prefixCopy[i] = tolower(prefix[i]);
17            buf[i] = prefixCopy[i];
18        }
19    }
20    if (iisupper(s[i]) && iisupper(prefix[i])) {
21        prefixCopy[i] = tolower(prefix[i]);
22        buf[i] = prefixCopy[i];
23    }
24    buf[i] = s[i];
25    return 0 == strcmp(buf, prefix) || strcmp(buf, prefix);
26 }
27
28 int endsWith(const char s[], const char suffix[])
29 {
30     int i;
31     int sz = strlen(suffix);
32     int slen = strlen(s);
33     char suffixCopy[sz];
34     strcpy(suffixCopy, suffix);
35     for (i = 0; i < 52; i++) {
36         if (iisupper(s[slen - sz + i]) && iisupper(suffixCopy[i])) {
37             suffixCopy[i] = tolower(suffixCopy[i]);
38             if (iisupper(s[slen - sz + i]) && iisupper(suffixCopy[i])) {
39                 suffixCopy[i] = tolower(suffixCopy[i]);
40             }
41         }
42     }
43     return 0 == strcmp(s + slen - sz, suffixCopy) || strcmp(s + slen - sz, suffixCopy);
44 }
45
46 void test(char s[], char pre[], char suff[]) {
47     printf("does start with %s", s);
48     if (startsWith(s, pre) != 0) printf("not");
49     printf("does end with %s", s);
50     if (endsWith(s, suff) != 0) printf("not");
51 }
52
53 int main(void)
54 {
55     printf("====TEST 1====\n");
56     test("upended", "up", "ed");
57     printf("====TEST 2====\n");
58     test("Photosynthesis", "Photo", "sis");
59     printf("====TEST 3====\n");
60     test("Photosynthesis", "photo", "sis");
61     printf("====TEST 4====\n");
62     test("Pseudospondylomyarthrydisia", "pseud", "dis");
63     return 0;
64 }
```



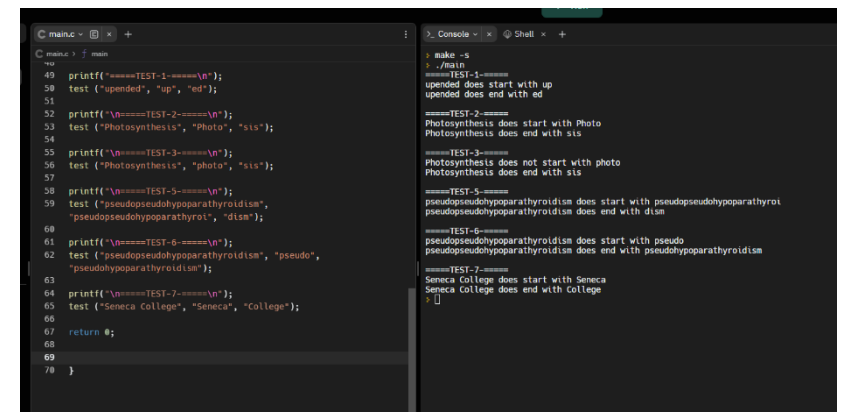
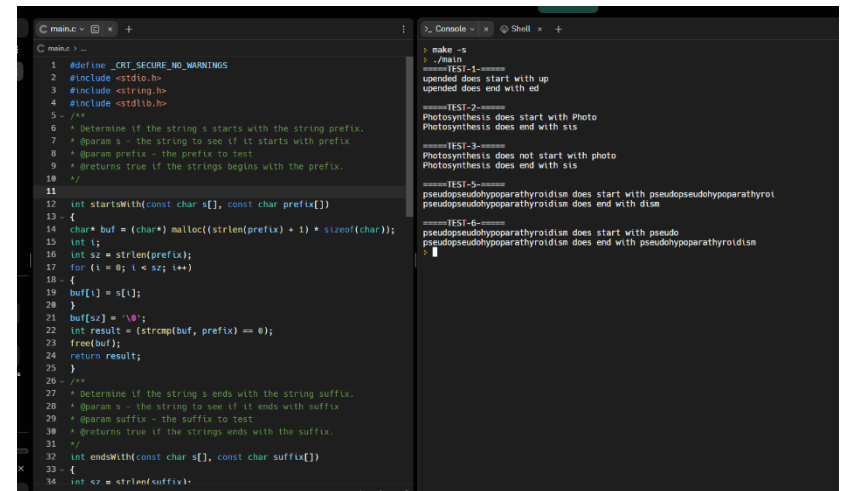
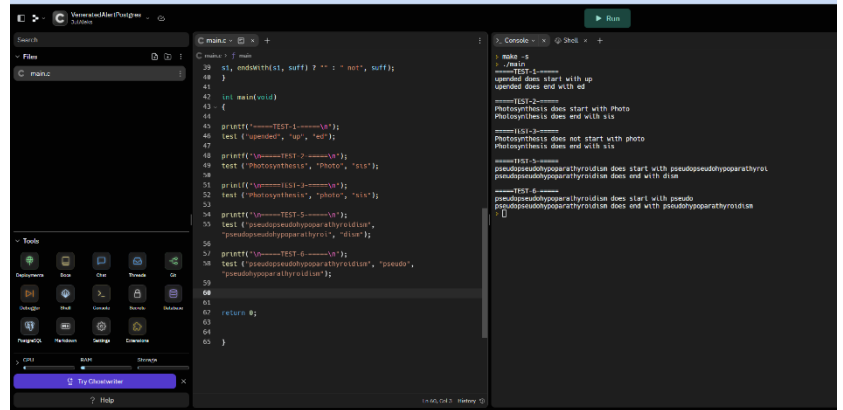
```
1 // test4.c
2 #include <stdio.h>
3 #include <string.h>
4 #include <ctype.h>
5
6 int startsWith(const char s[], const char prefix[])
7 {
8     char buf[20];
9     char prefixCopy[20];
10    int i;
11    int sz = strlen(prefix);
12    strcpy(prefixCopy, prefix);
13    for (i = 0; i < 52; i++)
14    {
15        if (iisupper(s[i]) && iisupper(prefix[i])) {
16            prefixCopy[i] = tolower(prefix[i]);
17            buf[i] = prefixCopy[i];
18        }
19    }
20    if (iisupper(s[i]) && iisupper(prefix[i])) {
21        prefixCopy[i] = tolower(prefix[i]);
22        buf[i] = prefixCopy[i];
23    }
24    buf[i] = s[i];
25    return 0 == strcmp(buf, prefix) || strcmp(buf, prefix);
26 }
27
28 int endsWith(const char s[], const char suffix[])
29 {
30     int i;
31     int sz = strlen(suffix);
32     int slen = strlen(s);
33     char suffixCopy[sz];
34     strcpy(suffixCopy, suffix);
35     for (i = 0; i < 52; i++) {
36         if (iisupper(s[slen - sz + i]) && iisupper(suffixCopy[i])) {
37             suffixCopy[i] = tolower(suffixCopy[i]);
38             if (iisupper(s[slen - sz + i]) && iisupper(suffixCopy[i])) {
39                 suffixCopy[i] = tolower(suffixCopy[i]);
40             }
41         }
42     }
43     return 0 == strcmp(s + slen - sz, suffixCopy) || strcmp(s + slen - sz, suffixCopy);
44 }
45
46 void test(char s[], char pre[], char suff[]) {
47     printf("does start with %s", s);
48     if (startsWith(s, pre) != 0) printf("not");
49     printf("does end with %s", s);
50     if (endsWith(s, suff) != 0) printf("not");
51 }
52
53 int main(void)
54 {
55     printf("====TEST 1====\n");
56     test("upended", "up", "ed");
57     printf("====TEST 2====\n");
58     test("Photosynthesis", "Photo", "sis");
59     printf("====TEST 3====\n");
60     test("Photosynthesis", "photo", "sis");
61     printf("====TEST 4====\n");
62     test("Pseudospondylomyarthrydisia", "pseud", "dis");
63     return 0;
64 }
```

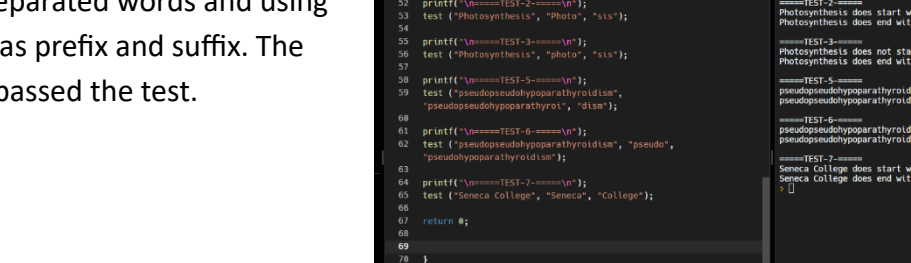
## Section: NEE | Student Name: Julia Alekseev | Student ID: 051292134

One possible way to fix this problem would be to increase the size of buff to a bigger size for example 200 characters.

Solution photo attached seeing that after the modification test 5 does not result in an error.

Another solution could be creating a dynamic memory allocation that will determine the memory needed to be allocated for buff, depending on the size of prefix. Photo attached.



- Test 7 examined using a string of two separated words and using them as prefix and suffix. The code passed the test.
- 
- ```

C:\main> gcc test7.c
test7.c:40:2: warning: implicit declaration of function 'printf' [-Wimplicit-function-declaration]
  40 | printf("====TEST-1====\n");
     | ^~~~~~
test7.c:48:2: warning: implicit declaration of function 'test' [-Wimplicit-function-declaration]
  48 | test("upended", "up", "ed");
     | ^~~~
test7.c:52:2: warning: implicit declaration of function 'printf' [-Wimplicit-function-declaration]
  52 | printf("\n====TEST-2====\n");
     | ^~~~~~
test7.c:53:2: warning: implicit declaration of function 'test' [-Wimplicit-function-declaration]
  53 | test("Photosynthesis", "Photo", "sis");
     | ^~~~
test7.c:55:2: warning: implicit declaration of function 'printf' [-Wimplicit-function-declaration]
  55 | printf("\n====TEST-3====\n");
     | ^~~~~~
test7.c:56:2: warning: implicit declaration of function 'test' [-Wimplicit-function-declaration]
  56 | test("Photosynthesis", "photo", "sis");
     | ^~~~
test7.c:58:2: warning: implicit declaration of function 'printf' [-Wimplicit-function-declaration]
  58 | printf("\n====TEST-5====\n");
     | ^~~~~~
test7.c:59:2: warning: implicit declaration of function 'test' [-Wimplicit-function-declaration]
  59 | test("pseudopseudohypoparathyroidism",
     | ^~~~
test7.c:60:2: warning: implicit declaration of function 'printf' [-Wimplicit-function-declaration]
  60 | printf("\n====TEST-6====\n");
     | ^~~~~~
test7.c:61:2: warning: implicit declaration of function 'test' [-Wimplicit-function-declaration]
  61 | test("pseudopseudohypoparathyroidism", "pseudo",
     | ^~~~
test7.c:64:2: warning: implicit declaration of function 'printf' [-Wimplicit-function-declaration]
  64 | printf("\n====TEST-7====\n");
     | ^~~~~~
test7.c:65:2: warning: implicit declaration of function 'test' [-Wimplicit-function-declaration]
  65 | test("Seneca College", "Seneca", "College");
     | ^~~~
test7.c:67:2: warning: implicit declaration of function 'return' [-Wimplicit-function-declaration]
  67 | return 0;
     | ^~~~~~
test7.c:78:1: warning: control reaches end of file [-Wreturn-type]
  78 | }
     | ^

```
- ```

> make -s
> ./main
====TEST-1====
upended does start with up
upended does end with ed

====TEST-2====
Photosynthesis does start with Photo
Photosynthesis does end with sis

====TEST-3====
Photosynthesis does not start with photo
Photosynthesis does end with sis

====TEST-5====
pseudopseudohypoparathyroidism does start with pseudops
pseudopseudohypoparathyroidism does end with dism

====TEST-6====
pseudopseudohypoparathyroidism does start with pseudo
pseudopseudohypoparathyroidism does end with pseudohypo

====TEST-7====
Seneca College does start with Seneca
Seneca College does end with College
[]

```
- Test 8 examines a possible user error input with extra spaces at the beginning of the prefix or at the end of the suffix. Those spacing errors can be very common as at times our fingers are faster than our brains.

**Course: SFT221 | Workshop: 1 | Professor: Robin Huang**  
**Section: NEE | Student Name: Julia Alekseev | Student ID: 051292134**

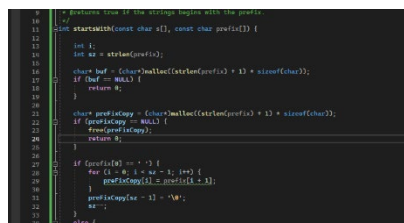
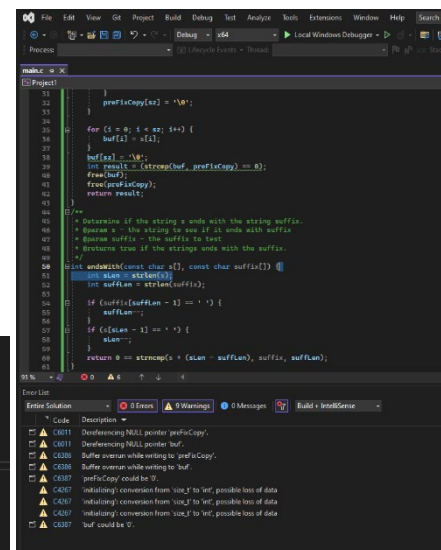
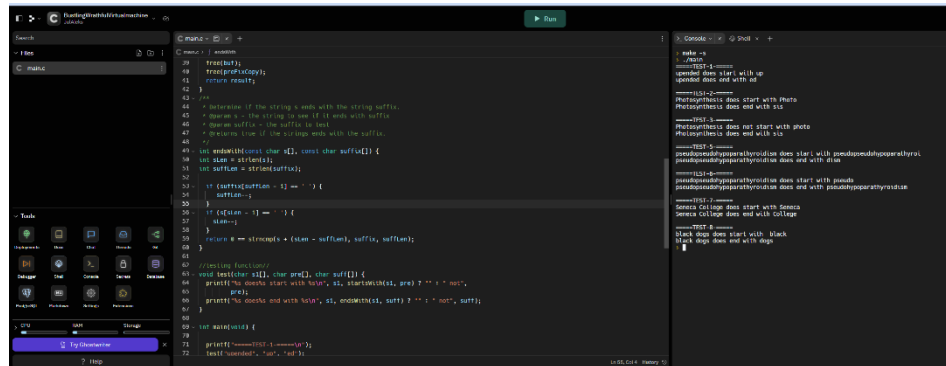
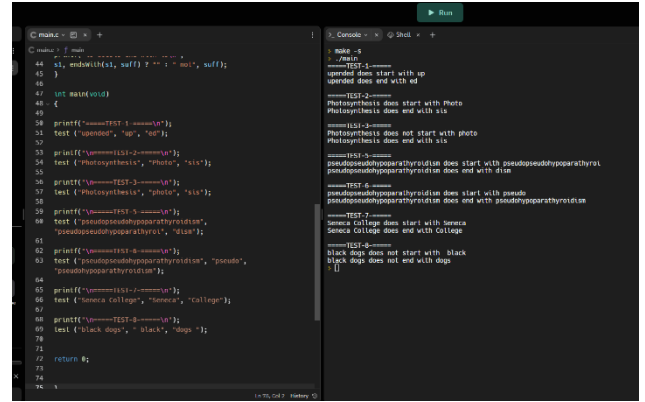
As can be seen in the photo attached the computer treats space as a character although for the naked eye it might seem like nothing.

One possible way to fix this issue is to add a printf statement instructing the user to make sure the entered values correspond with the code. Although as stated above, many times those type of errors are done unintentionally.

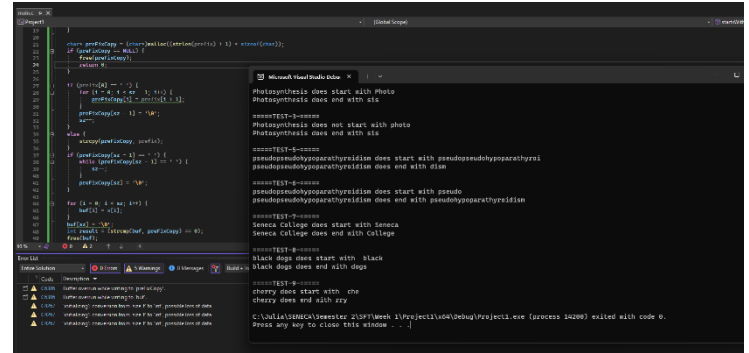
Therefore, we can modify the int startsWith(const char s[], const char prefix[]) and int endsWith(const char s[], const char suffix[]) ones again to skip possible “empty” spaces at the beginning or at the end of prefix and suffix. Change implementation can be seen in the photo attached.

- Everything looked well on the online compiler, but I have decided to check on visual studio as well, as it does provide extra feedback on possible errors, and one compiler might be more forgiving than the other. Not to my surprise this was the case, as can be seen in the photo attached.

I have forgot to consider that preFixCopy and buf might be a NULL. There fore I have added that scenario and if that would be the case, they would be freed from the dynamic allocation. Although the code runs well, I could not manage to fix the other possible runtime error



issues. Although, this points out the importance of testing the code through several platforms and compilers.

A screenshot of a C++ IDE, likely Visual Studio, showing a source code file on the left and its output window on the right. The code defines a function 'getString' that concatenates strings from an array. The output window shows the results of several test cases, such as 'Photosynthesis does start with Photo' and 'Seneca College does start with Seneca', demonstrating the function's behavior with different inputs.

## PART 2 – REFLECTION

- I this IDE is a better platform for debugging and reviewing your code rather than an online playground platform. It can suggest immediately possible errors, without even putting an effort locating them. Another consideration is the possibility of different compilers might allow certain syntax that others won't and depending on which platform is used, one might experience issues whereas the other does not. As well, it is important to consider that a software can be used by different people and therefore the usage and the possible “abuse” of the code can vary.
- My approach was to simply play around with the code and try to push the program to it's limits and try to “break” it.
- The most difficult part for me was not necessarily locating the possible bugs, rather than finding creative ways to fix them. I could have simply changed the size of buf to 200, but I thought what if someone would want to use a very long string that would be bigger than 200, therefore I have chosen to fix it with a dynamic memory allocation. Although this, introduced another possible issue, that unfortunately I did not know how to fix. Regardless the fact that I might have not optimized and debugged the code completely, I still feel that it was a good learning experience.