Test Cases
Refer to stringIdentifier-test-cases.xlsx

Bugs
1. Any of the imputed strings as NULL
2. When the evaluated string is bigger than the defined buffer in startsWith()
3. Any of the imputed strings is empty
4. When the prefix is bigger than the evaluated string
5. When the suffix is bigger than the evaluated string

Solution
To solve bugs related incorrect input (NULL, oversized, empty string)a new function called inputStingVerifier(char *str) was created. This functions receives a string and returns true or false if it fallows the input rules being not NULL, not bigger than max size and not empty. Additionally the function would prompt the user if it imputed an on valid type.

```
int inputStringVerifier(char *str)
{
   if (str == NULL)
   {
      printf("The string value nor pointer can be NULL.\n");
      return 0;
   }
   else if (strlen(str) <= 0)
   {
      printf("The string cannot be empty, you need at least 1 character.\n");
      return 0;
   }
   else if (strlen(str) > MAX_SIZE)
   {
      printf("The string cannot be bigger than %d characters.\n", MAX_SIZE);
      return 0;
   }
   else
   {
      return 1;
   }
}
```

In the scenario where the suffix or prefix size is bigger than the evaluated string, the main function was modified so it automayicaly returns false when suffix/prefix size is bigger.

```
void stringIdentifier(char *s1, char *prefix, char *suffix, int *result)
{
   result[0] = -1;
   result[1] = -1;
   if (inputStringVerifier(s1))
   {
      if (inputStringVerifier(prefix))
      {
         result[0] = 0;
         if (strlen(prefix) <= strlen(s1))
         {
            printf("|%s| does %s start with |%s|\n",
               s1, startsWith(s1, prefix) ? "\b" : "not", prefix);
            result[0] = startsWith(s1, prefix);
         }

      }

      if (inputStringVerifier(suffix))
      {
         result[1] = 0;
         if (strlen(suffix) <= strlen(s1))
         {
            printf("|%s| does %s end with |%s|\n",
               s1, endsWith(s1, suffix) ? "\b" : "not", suffix);
            result[1] = endsWith(s1, suffix);
         }
      }
   }
}
```

Reflextion

A reflection where you consider whether testing or inspection identified more bugs in this case. State why you think one way worked better than the other. How could you improve the technique that worked less well?

- ➔ The inspection process proved to be more effective in identifying bugs in this case. It allows to generate targeted test cases directly from the executed process. For instance, I discovered a bug related to string indexing while debugging the code and scrutinizing how it determined which part of the string to compare. The reason why the inspection process worked better than testing, thanks to its focused on specific areas where potential problems might arise and address them proactively.

To improve the testing technique, I would implement massive testing covering a wide range of scenarios. Additionally, employing automated testing tools or harnessing the power of unit testing frameworks could help enhance the testing process, making it more efficient in identifying bugs without relying solely on manual inspection.

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?

- ➔ I didn't have much trouble finding the obvious bugs tied to invalid inputs, as they are quite common and straightforward. However, when it came to identifying bugs related to the functionality of the program, it required a deeper dive into the code and understanding how it interacted with memory. The challenge lay in pinpointing the precise cause of the functionality issue, even though the symptoms were apparent.