KAIST
EE 209: Programming Structures for EE

Assignment 2: String Manipulation

(Acknowledgment: This assignment is borrowed and modified from Princeton's COS 217)

Purpose

The purpose of this assignment is to help you learn/review (1) arrays and pointers in the C programming language, (2) how to create and use stateless modules in C, (3) the "design by contract" style of programming, and (4) how to use the GNU/UNIX programming tools, especially `bash`, `emacs`, gcc, and gdb.

Rules

Implement the string functions listed by the Table in Part 1. Part 2 is the "on your own" part of this assignment, which is worth 50% of this assignment.

You will get an **extra** 10% of the full score if you implement the Part 1 only with pointer notation when you access the character. See the Extra credit section below (Extra Credit for Part 1).

Background

As you know, the C programming environment provides a standard library. The facilities provided in the standard library are declared in header files. One of those header files is string.h; it contains the declarations of "string functions," that is, functions that perform operations on character strings. Appendix D of the King textbook, Appendix B3 of the Kernighan and Ritchie textbook, and the UNIX "man" pages describe the string functions. The string functions are used heavily in programming systems; certainly any editor, compiler, assembler, or operating system created with the C programming language would use them.

Overview of Your Task

Your task in this assignment is to use C to create the "Str" module that provides string manipulation functions. Specifically, design your Str module so that each function behaves the same as described below. Your task in this assignment is twofold.

[Part 1] Read the description of the basic string library functions carefully, and implement each function. The basic functions are most commonly used standard string functions. Each function should behave the same as its corresponding standard C function.

[Part 2] Implement a simplified version of `grep` using `Str` functions. Read the provided file that contains skeleton code carefully, edit the file to make it process the required functionalities: find, replace, diff.

Part 1: The Basic `Str` Function Implementation

Your task for the first part is to use C to implement five basic string manipulation functions: `StrGetLength(), StrCopy(), StrCompare(), StrSearch(), StrConcat()`. Those five functions should follow the format of the corresponding standard C library functions. You can easily find the function's description and operation in the UNIX "man" page.

The following table shows the required basic string functions in Part 1 and their corresponding function names in the standard C library.

Str Function	Standard C Function	Man page link
StrGetLength	strlen	strlen man page
StrCopy	strcpy	strcpy man page
StrCompare	strcmp	strcmp man page
StrSearch	strstr	strstr man page
StrConcat	strcat	strcat man page

Use the `Str` module's *interface* in a file named `str.h`, and place your `Str` function definitions in `str.c`.

str.h str.c

Note that your Str functions **should not** call any of the standard string functions. In the context of this assignment, pretend that the standard string functions do not exist. However, your functions may call each other, and you may define additional (non-interface) functions.

Design each function definition so it calls the `assert` macro to validate the function's parameters. In that way, your `Str` functions should differ from the standard C string functions. Specifically, design each function definition to assert that each parameter is not `NULL`. See the note below for more information of the `assert()` macro function.

Beware of type mismatches. In particular, beware of the difference between type size_t and type int: a variable of type size_t can store larger integers than a variable of type int can. Your functions should (in principle) be able to handle strings whose lengths exceed the capacity of type int. Also beware of type mismatches related to the use of the const keyword.

Extra Credit for Part 1: Implement functions with pointer notation

There are various ways to implement the functions in Part 1. Especially, you can access the character by pointer dereferencing like `*pcSrc` or by using an array notation such as `pcSrc[uiLength]` .

Here are two examples of `StrGetLength()` implementation. The first code implements the StrGetLength() function with the array notation; it traverses each given string or accesses the character using an index relative to the beginning of the string. However, with the pointer notation, the second version traverses each given string using an incremented pointer.

```
size_t StrGetLength(const char pcSrc[]) /* Use array notation */
{
   size_t uiLength = 0U;
   assert(pcSrc != NULL);
   while (pcSrc[uiLength] != '\0')
      uiLength++;
   return uiLength;
}
```

```
size_t StrGetLength(const char *pcSrc) /* Use pointer notation */
{
   const char *pcEnd;
   assert(pcSrc != NULL);
   pcEnd = pcSrc;
   while (*pcEnd != '\0') /* note that *(pcSrc + uiLength) is valid but is "NOT" acceptable as pointer notation */
      pcEnd++;
   return (size_t)(pcEnd - pcSrc);
}
```

You can freely implement Part 1. However, **if you implement part 1 only with pointer notation, you can get extra 10% of the full score. Note: if you simply convert array notation to pointer notation (e.g., *(p+i) instead of p[i]), that does not count as pointer notation.** That is, pointer notation should not deal with an index. We will strictly apply the criteria for pointer notation, so please avoid using notation like (p+i) to get the full credit. p++/-- and *p are just fine. Please write your choice of implementation in the `readme` file. That is, specify if you used only the pointer notation for Part 1 in the `readme` file.

Test your Str Functions

We provide a test client (`client.c`) that compares the results of the `Str` and C standard library functions with various input. Please use this code for testing and debugging. You can compile the test client with `gcc209` as follows.

`client.c`

```
gcc209 -o client client.c str.c
```

Then, test each of your `Str` functions separately by providing a function name as an argument. For example,

```
./client StrCopy
```

will test `StrCopy`. Actually, the client accepts any one of the `Str` function names as a command-line argument:

```
./client [StrGetLength|StrCopy|StrCompare|StrSearch|StrConcat]
```

Note that passing all tests provided by the client does **not** mean that your function always behaves correctly. Please devise your own testing (e.g., by changing the client code) for more confidence. Note that we may use a different test client for grading.

Part 2: Simple Grep

`grep` is a popular UNIX tool that manipulates input strings. In this part, you implement a simplified version of `grep` called `sgrep`. `sgrep` provides these three functionalities.

- **Find (`-f search-string`)**: reads each line from standard input (`stdin`) and prints out **only the lines** that contain `search-string` to standard output (`stdout`).
- **Replace (`-r string1 string2`)**: reads each line from standard input (`stdin`), **replaces all occurrences** of `string1` with `string2`, and prints it out to standard output (`stdout`). If `string1` is not found in the line, the original line is copied to `stdout`.
- **Diff (`-d file1 file2`)**: compares the two files (file1 and file2) line by line, and prints out **only the different lines** with their line numbers to standard output (`stdout`). The output format of the different line should be as follows:

  ```
  file1@line-num:file1's line
  file2@line-num:file2's line
  ```

Here are some usage examples of `sgrep` (you can also see the test files (`google_wiki.txt` and `microsoft.txt`) which are used in the following example):

```
$ ./sgrep -f Google < google_wiki.txt
Google Inc. is an American multinational technology company specializing
Google was founded by Larry Page and Sergey Brin while they were Ph.D.
 supervoting stock. They incorporated Google as a privately held company on
 be evil".[9][10] In 2004, Google moved to its new headquarters in Mountain
View, California, nicknamed the Googleplex.[11] In August 2015, Google
Alphabet Inc. When this restructuring took place on October 2, 2015, Google
became Alphabet's leading subsidiary, as well as the parent for Google's

$ ./sgrep -r Google Microsoft < google_wiki.txt > microsoft.txt
$ cat microsoft.txt (output is abbreviated with ...)
Microsoft Inc. is an American multinational technology company specializing
...
Microsoft was founded by Larry Page and Sergey Brin while they were Ph.D.
...
 supervoting stock. They incorporated Microsoft as a privately held company on
...
 be evil".[9][10] In 2004, Microsoft moved to its new headquarters in Mountain
...
View, California, nicknamed the Microsoftplex.[11] In August 2015, Microsoft
...
Alphabet Inc. When this restructuring took place on October 2, 2015, Microsoft
...
became Alphabet's leading subsidiary, as well as the parent for Microsoft's

$ ./sgrep -d google_wiki.txt microsoft.txt
google_wiki.txt@1:Google Inc. is an American multinational technology company specializing
microsoft.txt@1:Microsoft Inc. is an American multinational technology company specializing
google_wiki.txt@6:Google was founded by Larry Page and Sergey Brin while they were Ph.D.
microsoft.txt@6:Microsoft was founded by Larry Page and Sergey Brin while they were Ph.D.
google_wiki.txt@9: supervoting stock. They incorporated Google as a privately held company on
microsoft.txt@9: supervoting stock. They incorporated Microsoft as a privately held company on
google_wiki.txt@15: be evil".[9][10] In 2004, Google moved to its new headquarters in Mountain
microsoft.txt@15: be evil".[9][10] In 2004, Microsoft moved to its new headquarters in Mountain
google_wiki.txt@16:View, California, nicknamed the Googleplex.[11] In August 2015, Google
microsoft.txt@16:View, California, nicknamed the Microsoftplex.[11] In August 2015, Microsoft
google_wiki.txt@18:Alphabet Inc. When this restructuring took place on October 2, 2015, Google
microsoft.txt@18:Alphabet Inc. When this restructuring took place on October 2, 2015, Microsoft
google_wiki.txt@19:became Alphabet's leading subsidiary, as well as the parent for Google's
microsoft.txt@19:became Alphabet's leading subsidiary, as well as the parent for Microsoft's
```

`google.txt` `microsoft.txt`

Rules:

- **General rules**
  - Use your Str functions to implement the functionality. That is, you need to finish Part 1 first to get any credit for this task. You should not use any string function in string.h
  - Assume each line (including a new line character ('\n') is no more than 1023 bytes. You should stop your program with a proper error message if you encounter a line larger than 1023 bytes.
  - A string argument (`search-string, string1, string2`) refers to a sequence of any non-space characters. It can be enclosed with double quote(") characters and can be either empty ("") or can include a space (e.g., "hello world"). No need to handle an escape character for this assignment.
  - Assume a string or a file argument is no more than 1023 bytes. You should stop your program with a proper error message if you encounter a command-line argument that's too long.

    - Any error message should be printed out to standard error (`stderr`). Use `fprintf(stderr, ...);` for that.
    - Unlike Part 1, you don't have to add assert function while implementing sgrep. In other words, after printing out the error messages in error case, sgrep should be finished with EXIT_FAILURE as the return value. Details are written in skeleton code, so please read the comments in skeletion code (`sgrep.c`).
- **Rules for Find and Replace**
    - `string1` cannot be an empty string. In such a case, your program should stop with a proper error message (e.g., "Error: Can't replace an empty substring").
    - Other string arguments (`string2, search-string`) **can** be an empty string. That is, an empty search string matches any line, and if `string2` is an empty string, it removes any occurrences of `string1` in the matching line.
    - **New:** Do not use dynamic memory allocation (malloc(), calloc(), realloc(), alloca(), etc.) nor a large local array. Note that it suffices to delcare a char buffer whose size is 1024 bytes (or 1025 bytes as shown in sgrep.c) even for string replacement. Any char buffer larger than 1025 bytes is considered too large for the assignment.

Tips:

- We provide a skeleton code file (`sgrep.c`). You can start with the file.
- `fgets` should be useful for reading a line from a file. 'man fgets' should give you more information.
- (Added:) We provide a solution binary. You can download and run a solution binary by executing the commands below:

```
$ wget http://ee209.kaist.ac.kr/assignment/string/file/sgrep_sol
$ chmod +x sgrep_sol
```

`sgrep.c` `sgrep_sol`

Logistics

Develop on lab machines using `emacs` to create source code and `gdb` to debug.

For part 1, you implement the code `str.c` and `str.h`, which should contain the definitions of required functions.

For part 2, a skeleton C file is available here: sgrep.c. It implements the basic I/O and argument parsing, so all you need is fill out the rest of the functionalities with `Str` functions. Feel free to use this file as your starting point (of course, you don't have to use it), and change any part in the code if you'd like.

Create a `readme` text file that contains:

- Your name and your student ID
- A description of whatever help (if any) you received from others while doing the assignment, and the names of any individuals with whom you collaborated, as prescribed by the course "Policy" web page.
- *Is it possible for `StrCopy` to call `assert` to verify that the destination memory area specified by the caller is large enough? Explain.*
- (Optionally) Please write the your implementation method in `readme` text file whether you use pointer notation for the extra credit.
- (Optionally) An indication of how much time you spent doing the assignment.
- (Optionally) Your assessment of the assignment.
- (Optionally) Any information that will help us to grade your work in the most favorable light. In particular you should describe all known bugs.

Submission

Use KAIST KLMS to submit your assignments. Your submission should be one gzipped tar file whose name is `YourStudentID_assign2.tar.gz`

For example, if your student ID is 20191234, please name the file as `20191234_assign2.tar.gz`

Create a local directory named '`YourStudentID_assign2`' and place all your files in it. Then, `tar` your submission file. Please refer here for how to archive your assignment. **Do not archive your submission file without creating a directory and moving your files inside.**

Your submission need to include the following files:

- (Task 1) `str.c` and `str.h` files
- (Task 2) updated `sgrep.c` file
- `readme` text file
- document of Ethics. Sign on the document, save it into a PDF file, and submit it.

Ethics document

Your submission file should look like this:

```
20191234_assign2.tar.gz
    20191234_assign2
        str.c
        str.h
        sgrep.c
        readme
        EthicsOath.pdf
```

Do not add files which has not been mentioned above.
Do not put another source files in your submission. Your implementation must lay in `str.c`, `str.h`, and `sgrep.c`, not in other files.

The name of the files must be exact. Note that file name is case sensitive. You might not get the full credit if the names are not correct.

Grading

If your code cannot be compiled at `eelab5` with `gcc209`, we cannot give you any points. Please double check before you submit.

We will grade your work on quality from the user's point of view and from the programmer's point of view. To encourage good coding practices, we will deduct points if `gcc209` generates warning messages.

From the user's point of view, your module has quality if it behaves as it should.

In part, style is defined by the rules given in *The Practice of Programming* (Kernighan and Pike), as summarized by the Rules of Programming Style document. These additional rules apply:

Names: You should use a clear and consistent style for variable and function names. One example of such a style is to prefix each variable name with characters that indicate its type. For example, the prefix `c` might indicate that the variable is of type `char`, `i` might indicate `int`, `pc` might mean `char*`, `ui` might mean `unsigned int`, etc. But it is fine to use another style -- a style which does not include the type of a variable in its name -- as long as the result is a readable program.

Line lengths: Limit line lengths in your source code to 72 characters. Doing so allows us to print your work in two columns, thus saving paper.

Comments: Each source code file should begin with a comment that includes your name, the number of the assignment, and the name of the file.

Comments: Each function should begin with a comment that describes what the function does from the caller's point of view. The function comment should:

- Explicitly refer to the function's parameters (by name) and the function's return value.
- State what, if anything, the function reads from standard input or any other stream, and what, if anything, the function writes to standard output, standard error, or any other stream.
- State which global variables the function uses or affects.
- Appear in both the interface (.h) file for the sake of the *clients* of the function and the implementation (.c) file for the sake of the *maintainers* of the function.

Parameter Validation: Validate function parameters via asserts whenever possible.

Note: assert and NDEBUG

assert is a macro implementations of assertion, used for verifying the conditions. If the condition is true, it does nothing. However, if the conditions is FALSE, it displays an error messages and aborts the running program.

In this assignment, you should validate the function's parameters with `assert`. When you try to check whether your `Str` functions validate the given parameters correctly or not, the aborted program may annoy you. In that case, you can add `NDEBUG` macro in your source file to ignore the assert functions. Otherwise, you can also add -D NDEBUG argument to the `gcc`. The `-D NDEBUG` argument commands `gcc209` to define the `NDEBUG` macro, just as if the preprocessor directive `#define NDEBUG` appeared in the specified .c file(s). Following OBcommands are example of disabling assert using -D option

```
gcc209 -D NDEBUG sgrep.c str.c -o sgrep
```

If NDEBUG is defined as a macro name in the source file, the assert macro is defined as `((void)0)`, which means that the assert macro will be ignore.

Note: Using Idioms

C programmers sometimes use idioms that rely on the fact that the null character ('\0'), the `NULL` address, the integer 0, and the logical concept FALSE have the same representation. You may use those idioms. For example, you may define `StrGetLength` like this:

```
size_t StrGetLength(const char pcSrc[])
{
   size_t uiLength = 0U;
   assert(pcSrc); /* NULL address, 0, and FALSE are identical. */
   while (pcSrc[uiLength]) /* null character and FALSE are identical. */
      uiLength++;
   return uiLength;
}
```

or like this:

```
size_t StrGetLength(const char *pcSrc)
{
   const char *pcEnd;
   assert(pcSrc); /* NULL address, 0, and FALSE are identical. */
   pcEnd = pcSrc;
   while (*pcEnd) /* null character and FALSE are identical. */
      pcEnd++;
   return (size_t)(pcEnd - pcSrc);
}
```

But you are not required to use those idioms. In fact, we recommend that you avoid the use of idioms that adversely affect understandability.

Note: The `const` Keyword and `StrSearch`

The use of the `const` keyword within `StrSearch` is tricky, as this question-and-answer sequence indicates.

Question

According to the man pages, the parameters of `strstr` are of type `const char*`. That implies that the parameters of `StrSearch` also should be of type `const char*`. Why are they not of type `char*`?

Answer

Suppose you were to define `StrSearch` like this:

```
char *StrSearch(char *pcHaystack, char *pcNeedle)
{
   ...
}
```

Further suppose the client then calls `StrSearch` like this:

```
const char *pcBig = "hello";
const char *pcSmall = "lo";
 ...
 ... StrSearch(pcBig, pcSmall) ...
 ...
```

(Note that's a perfectly reasonable way to call the function.) In that case the compiler, noting that `pcBig` is of type `const char*` and that `pcHaystack` is of type `char*`, would generate a warning on the function call. Thus `pcHaystack` (and `pcNeedle`) should be of type `const char*`.

Question

According to the man pages, the return type of `strstr` is `char*`. That implies that the return type of `StrSearch` also should be of type `char*`. Why is the return type not `const char*`?

Answer

Suppose you were to define `StrSearch` like this:

```
const char *StrSearch(const char *pcHaystack, const char *pcNeedle)
{
   ...
}
```

Further suppose the client then calls `StrSearch` like this:

```
char *pcBig = "hello";
char *pcSmall = "lo";
char *pc;
 ...
pc = StrSearch(pcBig, pcSmall);
 ...
```

(Note that's a perfectly reasonable way to call the function.) In that case the compiler, noting that `pc` is of type `char*` and that the value returned by `StrSearch` is of type `const char*`, would generate a warning on the assignment statement. Thus the return type of `StrSearch` should be `char*` and should not be `const char*`.

Question

Within the definition of `StrSearch`, I decided to define a local variable named `pc` that "points into" the first of the two given strings, as indicated by this code:

```
char *StrSearch(const char *pcHaystack, const char *pcNeedle)
{
   ...
   pc = pcHaystack;
   ...
   /* Increment pc so it points to the appropriate character. */
   ...
   return pc;
}
```

If I define `pc` to be of type `char*`, then the assignment statement generates a warning. If I define `pc` to be of type `const char*`, then the `return` statement generates a warning. How can I resolve that problem?

Answer

Unfortunately, C provides no elegant solution. We recommend that you define `pc` to be of type `const char*` so the assignment statement is warningless. Then use a cast operator in the `return` statement:

```
return (char*)pc;
```

to explicitly inform the compiler to not generate a warning. C programmers refer to that solution as "casting away the constness" of the variable. Sadly, often that inelegant technique is unavoidable.

FAQ

Question 1

The `StrGetLength` function is already implemented in the given file. So should we just leave it and finish other functions?

Answer

You should not implement the `StrGetLength` function.

Question 2

Do we have to replace the based on words or letters? For example, lets say the input file contains '`a car`' and let string 1 is 'a' and string 2 is 'e'. Which output is correct; '`e car`' or '`e cer`'?

Answer

'`e cer`' is correct. You have to scan all string occurrences (not words).

Question 3

In assignment 2, do we have to implement `strncpy`, `strncmp` as well?

Answer

You only need to implement `strcpy`, `strcmp`.

Question 4

I am confused with following question in readme. "Is it possible for StrCopy to call assert to verify that the destination memory area specified by caller is large enough? Explain" Is the question asking about my own Strcopy implementation or about Strcopy in general?

Answer

It is asking you to answer if the following question is theoretically possible in general.

Question 5

For part2 replace, it states that maximum buffer size is 1025 bytes. Do we have to consider the error case when the string becomes over 1025 byte size after replacement?

Answer

Note that it suffices to declare a char buffer whose size is 1024 bytes (or 1025 bytes as shown in `sgrep.c`) even for string replacement. Any char buffer larger than 1025 bytes is considered too large for the assignment.

Question 6

When implementing diff of sgrep, what error message should we print if one of the file is empty?

Answer

~~You MUST NOT print different error message.~~

You MUST NOT print any messages.

Question 7

According to general rules, it says that we should return `EXIT_FAILURE` for error cases. However, in skeleton code for `sgrep.c` it returns `FALSE` for error cases. Should we modify the skeleton code?

Answer

For each function it returns `FALSE` for error cases as you said. However the main function returns `EXIT_FAILURE` if the sub functions (e.g, find, diff replace) returns `FALSE`. Therefore, the main code will return `EXIT_FAILURE` as intended.

Question 8

On DoDiff, if one the file ends earlier than the other one, do we have to print DoDiff result until one file ends earlier and print the error message?

Answer

You have to print the DoDiff result until one file ends earlier.

Question 9

It is mentioned that we need to print error message '`failed to open [filename]`'. If both two files are unable to be opened, do we have to print the error message twice?

Answer

~~You only need to consider error message in terms of first file input.~~ That is, you only need to print error message once, while checking file inputs.

Question 10

To print "`Error: failed to open file [filename]\n`" message for file google_wiki.txt, which error message is correct?

1. `Error: failed to open file google_wiki.txt`
2. `Error: failed to open file [google_wiki.txt]`

Answer

1 is correct.

Question 11

When we read file using fgets, it reads the newline character for each line in the file, and therefore if we print this, it automatically produces newline. However, for the last line in the file, there is no newline. Let's say we are comparing two files that have different last line respectively, and we need to print the result. Should we include newline to our printed result?

Answer

You have to print the text with new line character just for last line. Because there always is a new line character in the end except last line.

Question 12

Do we have to consider the case that the replaced line has more than 1023 characters?

Answer

There were a lot of questions about error message when the length of replaced line has more than 1023 characters. You don't need to consider the case that the replaced line has more than 1023 characters, and it is also okay if you have already implemented the case.

Question 13

If I have two following files,

- 1st line of file1.txt: `abc\n`
- 1st line of file2.txt: `abc`

Do I need to consider these two files as different files?

Answer

For this case, since both file has same 1st line and file2 has no 2nd line, you need to print error message that file has ended early.

Question 14

If the search-string is nothing but blank (i.e.,''), what should we print for result?

Answer

You need to print every line.