String Functions

CS 580U – Spring 2024

Thursday Mar 28 11:59:59 PM EDT 2024

1 Instructions

1.1 Before You Begin

You can write your code anywhere on any machine, however it will be tested on a Github Actions runner compatible with `remote.cs.binghamton.edu`. Although you will get immediate output from the actions run, if your code fails on the runner you can request me to hand grade it on `remote` at no penalty. If it fails on the runner and on `remote.cs.binghamton.edu`, it is considered non-functioning *even if it runs on your personal computer*. Additionally, there is no implied claim that the provided code would even run on your personal machine. For example, the provided library **libsolutions.a** was compiled for the x86_64-unknown-linux-gnu target with GNU libc 2.36, and the **test** program expects to link with a system-installed **libcunit** 2.1-3 library and headers. Therefore it won't run on MacOS/Windows operating systems, on ARM CPUs, or even on another Linux computer if you haven't installed all libraries. These prerequisites are all satisfied for you on `remote.cs.binghamton.edu`.

1.2 Testing Your Code

In addition to implementing the functions in **functions.c**, you may also create a standalone program in a file such as **main.c** (not provided) which calls your functions in an system test of your design. But ensure all graded functionality is implemented exclusively within **functions.c**, and do not include a `int main(...)` function there. The Makefile includes a target `make test` which will build an executable linked to the CUnit library (see **??**). You can then run this executable using the command `./test`, which will give you an output like that of **??**. We will use this same command for grading in a Github Action.

1.3 Submitting Your Program

If you believe the Github grading runner has successfully graded your work:

1. Ensure that all of your work has been committed and the local repository has been pushed to the remote repository.

2. Copy the output from `git rev-parse --short HEAD` to get the SHA hash of HEAD

3. Paste the output into the Brightspace assignment submission page as a comment. You do not need to upload any files, I will verify the hash matches the hash on the runner.

If you believe the Github grading runnner has *unsuccessfully* graded your work or you wish to make a *late submission*:

1. Run `make submit` which will generate a .tar archive with the assignment name and your username.

2. Add "Manual grading requested" as a comment to the submission

3. Upload the archive to the Brightspace assignment submission page.

4. Your submission date will be used for grading.

1.4 Grading

The Makefile includes a target `make check` which will build the test executable and run it. Results will be both printed to the screen by the executable, and recorded to a file called **feedback.log** by the Makefile. You will receive points according to the following rubric:

Task	Points	Description
testNO_STRING_H	0	For grader use only
testMYSTRLEN	20	+1 per test case
testMYSTRCMP	20	+1 per test case
testMYSTRSTR	20	+1 per test case
testMYSTRSTRIP	20	+1 per test case
testANYALL	20	+1 per test case
Late Penalty	-	-5 first day, -10 subsequent days
Coding Standards	-	Up to -10 at grader discretion

Table 1: Grading rubric for the String Functions assignment.

testNO_STRING_H checks for use of prohibited header files and/or functions. If you fail this test, the grader will inspect for

"Coding Standards" means the structure, readability, and style either enhance or reduce the grader's ability to grade your submission. In general readable programs, following a consistent coding style, with meaningful variable/function/macro names, and sufficient comments to illustrate the programmer's intent will be awarded full credit. I encourage you to use tools such as linters, beautifiers, and documentation generators to improve your code quality. We will discuss some of these tools at a later time.

2 Questions

Implement the following functions in **functions.c** using prototypes located in **functions.h**:

1. `size_t mystrlen(const char* str);`
2. `int mystrcmp(const char *lhs, const char *rhs);`
3. `char *mystrstr(const char* str, const char* substr);`
4. `char *mystrstrip(const char* str);`
5. `bool any(const void *ptr, size_t nmemb, size_t size,bool (*pred)(const void *));`
6. `bool all(const void *ptr, size_t nmemb, size_t size,bool (*pred)(const void *));`
7. Ten predicate functions from `<ctype.h>`: `bool myisalnum(const void *ptr);`, `bool myisalpha(const void *ptr);`, etc.

2.1 String Length

Return the length of the given null-terminated byte string, that is, the number of characters in a character array whose first element is pointed to by `str` up to and not including the first null character. Note that a string may contain a non-terminating null character (e.g. the second test case below). You may not use any functions in `<string.h>`.

`str`	`mystrlen()`
"Binghamton$_{\%_t}$"	10
"Bing$_{\%_t}$hamton"	4

Table 2: Example inputs and outputs for `mystrlen()`

2.2 String Comparison

Compare two null-terminated byte strings lexicographically:

- if `lhs` appears before `rhs`, return -1,
- if `rhs` appears before `lhs`, return 1,
- otherwise, `lhs` and `rhs` are equal, and return 0.

Note that a prefix is always lexicographically sorted *before* the string it prefixes (e.g. fourth test case below). Also note that capital letters appear *before* all lowercase letters (in other words compare their ASCII values, e.g. fifth test case below). You may not use any functions in `<string.h>`.

lhs	rhs	mystrcmp()
apple$_{\%L}$	banana$_{\%L}$	-1
banana$_{\%L}$	apple$_{\%L}$	1
apple$_{\%L}$	apple$_{\%L}$	0
apple$_{\%L}$	apple$\sqcup_{\%L}$	-1
apple$_{\%L}$	BANANA$_{\%L}$	1

Table 3: Example inputs and outputs for `mystrcmp()`

2.3 String Substrings

Find the first occurrence of the null-terminated byte string pointed to by substr in the null-terminated byte string pointed to by str. Do not compare the terminating null characters. When found, return a pointer to the first character of the found substring, or a null pointer if the substring isn't found. Note that a substring may appear multiple times within the string. If this is the case, return the first appearance of the substring within the string (e.g. fourth test case below). You may not use any functions in `<string.h>`.

str	substr	mystrstr()
chocolate$_{\%L}$	choc$_{\%L}$	str
chocolate$_{\%L}$	hocol$_{\%L}$	str+1
chocolate$_{\%L}$	vanilla$_{\%L}$	NULL
chocolate$_{\%L}$	o$_{\%L}$	str+2

Table 4: Example inputs and outputs for `mystrstr()`

2.4 String Stripping

Return a pointer to a heap allocation of the minimum size necessary to contain the input string with leading and trailing whitespace removed. Whitespace is defined as one of the following characters:

- Horizontal tab (`0x09`, `'\t'`),
- Line feed (`0x0a`, `'\n'`),
- Vertical tab (`0x0b`, `'\v'`),
- Form feed (`0x0c`, `'\f'`),
- Carriage return (`0x0d`, `'\r'`),
- Space (`0x20`, `' '`)

The original string must remain unmodified, and whitespace that is not leading or trailing is not stripped. If the input string is a null pointer, return a null pointer and do not allocate memory. It is the caller's responsibility to check

for a returned null pointer and to free the allocation of the returned string. You may not use any functions in <string.h>

str	mystrstrip()	size of allocation
"chocolate"	"chocolate"	10
" chocolate "	"chocolate"	10
"\t\tchocolate\n"	"chocolate"	10
"\t\tchoco late\n"	"choco late"	11

Table 5: Example inputs and outputs for mystrstrip()

2.5 String Predicates

Given a pointer to a buffer, the number of members in that buffer, the size of each member, and a predicate function, return whether *any* or *all* of the members return true when the predicate is applied to each member. Each predicate function takes a const void* pointer, and returns a boolean true or false. In this specific assignment, we will write 10 wrapper functions for our predicates, each identical in nature, calling one of the character classification functions from <ctype.h>. For example, bool myisalnum(const void *ptr); is a wrapper for int isalnum(int ch); These wrappers are declared in **functions.h:95-211**, and their implementations can be as short as one line each if you are clever!

Note that for backwards compatibility with early architectures like the PDP-11, the character classification functions expect an int (not a char) as input and return an int as output. However, we need a void pointer as input and return a boolean as output. Therefore, you will need to figure out which casting conversions will be necessary – you should not have any compiler warnings from these casts if you do it correctly. Although you may not use any functions in <string.h>, you are expected to include and call the classification functions from <ctype.h>.

ptr	nmemb	size	pred	all()	any()
pointer to string "binghamton"	10	sizeof(char)	&myislower	true	true
pointer to string "Binghamton"	10	sizeof(char)	&myislower	false	true
pointer to string "BINGHAMTON"	10	sizeof(char)	&myislower	false	false

Table 6: Example inputs and outputs for mystrstrip()