Programming Assignment #3

This assignment will help you get acquainted with the basics of C strings, malloc, free, pointers, and structs. This is the third assignment that will be for a grade. Please take the time to read through the instructions and complete the function below.

Setup

Begin by downloading the starter files provided on Canvas and creating a new project on CLion.

UT Strings

In this project, we are creating an Abstract Data Type (ADT) to improve upon the standard C way of representing strings. We will use malloc and free, and get some practice with pointers, structs, and macros.

It is your job to implement UT String ADT. Several functions and function headers are already written for your convenience. You must use **without modifying** the `UTString` struct that is defined inside UTString.h. The struct consists of the following parts:

- `length`: the number of characters in the string (not including the null character or anything past it).
- `capacity`: the length of the longest string that can be stored.
- `string`: a pointer to where the string is stored. **The string must be allocated separately from the `UTString` itself.**
- `check`: the signature (~0xacedfade) stored after a string. It signifies that the object is in fact a `UTString`. We will check the value every time when working with a `UTString` to make sure no buffer overflow has occurred. To write the signature into your string, use `CHECK(s) = SIGNATURE`.

Example of a `UTString`:

`length` = 11
`capacity` = 11
`string` = 'Hello World'`\0~`0xacedfade

Functions

For this lab, you will implement the following functions:

- `UTString* utstrdup(const char* src)`: Create a UTString on the heap that holds a copy of source, setting the length, capacity, and check appropriately. Return a pointer to the UTString. Initally set the capacity equal to the length of the string.
- `UTString* utstrrev(const UTString* src)`: Reverses the `string` up to its length and returns `src`. **Only the contents of the string before the null terminator should reverse. The null terminator and the check should remain in the same position after the reversal.**
- `UTString* utstrcat(UTString* s, const char* suffix)`: Append characters to `s` from `suffix` until out of capacity or done copying. Do not copy more than can be stored. Do not allocate further space. Do use a null terminator and update the check. Return `s` after appending.

- `UTString* utstrcpy(UTString* dst, const char* src)`: Should replace characters in `dst` with characters from `src` until out of capacity or done copying. Do not copy more than can be stored. Do not allocate further space. Do use a null terminator and update the check. Return `dst` after copying
- `void utstrfree(UTString* self)`: Frees `self`. Must deallocate both `string` and the `UTString` itself.
- `UTString* utstrrealloc(UTString* s, uint32_t new_capacity)`: Used to reallocate space for `s`. If `new_capacity` is larger than the current `capacity`, create a buffer with `new_capacity` capacity, copy all the old contents, and deallocate the old buffer. Otherwise, do nothing. Either way, return `s` afterwards.

Bounds and Requirements

`length` should always be $\leq$ `capacity`.

If `capacity > length`, the bytes after 0xef are unknown.
Example:

`length` = 7
`capacity` = 11
`string` = 'shorter'`\0~`0xacedfade????

UTStrings must be stored on the heap. UTStrings will only be created by calling `utstrdup`. If the `check` is incorrect, your program should fail an assert and crash immediately. To check for this, call `bool isOurs(const UTString* s)` that is already written for you. Use `assert(isOurs(<UTString>))` to have your program fail the assert and crash properly.

Submission

Submit only your PA3.cpp file to Gradescope. **Please do not edit the name of this file**; the grader will give you a 0. After a short time, a few test case results should be visible to you on Gradescope. These are sample test cases, letting you know your Gradescope submission compiled and ran successfully on those tests. **Please note passing the sample cases DOES NOT MEAN you scored a 100 on the assignment.** Debug, debug, debug, and test, test, test!

Please do your own work on the project and do not share your code with others.

Good luck and have fun!

FAQ

Q1: Can we use String.h?
A: Yes, string.h is allowed for this lab

Q2: How evil are our users?
A: Quite malicious. Make sure to account for edge cases as specified

Workshop

A reminder that we host an assignment workshop the Friday before each assignment is due. If you ever need help with your assignment, feel free to stop by! Your TAs are always willing to help and we want you to succeed in the class!