Project Extra

Due December 9, 2019 at 11:59 PM

This project specification is subject to change at any time for clarification. For this extra credit project you will be working alone. You are tasked with writing a few functions that manipulate a safer version of C style strings. You will be allocating space for the “SafeString” (notice they are in quotes because these strings are just safer, not safe). The memory allocated for a “SafeString” must be the power of two that will be large enough to hold the requested size plus overhead. A “SafeString” will be a minimum of 64 bytes. Below is a description of the functions and their behavior.

```
// Returns a non-zero value if the string is a "SafeString"
char SafeStringIsValid(const char *str);

// Returns a pointer to the "SafeString" that can hold at least
// maxlen characters.
char *SafeStringAllocate(size_t maxlen);

// Returns a pointer to the "SafeString" that can holds a copy
// of str.
char *SafeStringFromCstr(const char *str);

// Frees of the memory for the "SafeString", if it is a valid
// "SafeString" otherwise the function does nothing.
void SafeStringDestroy(const char *str);

// Returns the maximum length that the "SafeString" can hold,
// returns 0 if str is not a "SafeString".
size_t SafeStringMaxLength(const char *str);

// Returns the length of the "SafeString", returns 0 if str is
// not a "SafeString".
size_t SafeStringLength(const char *str);

// Copies the contents of src into dest if dest is large enough
// to hold src. Returns dest on success. NULL is returned if
// dest is not a "SafeString", or there is insufficient room to
// hold src.
char *SafeStringCopy(char *dest, const char *src);

// Concatenates the contents of src onto the end of dest if dest
// is large enough to hold the concatenation. Returns dest on
// success. NULL is returned if dest is not a "SafeString", or
// there is insufficient room to hold the concatenation.
char *SafeStringConcatenate(char *dest, const char *src);
```

```
// Resizes the "SafeString" that is pointed to by *str to the
// size specified by newsize. If the *str does not point to a
// "SafeString", then 0 is returned. The returned size will be
// greater than or equal to newsize upon success, and *str will
// point to the newly allocated space if required.
size_t SafeStringResize(char **str, size_t newsize);
```

Important assumptions and requirements:

- You may only modify the `SafeString.c` file as it is the only one you will be turning in.
- You may only use the C standard library functions.
- Your code may not have memory leaks, it will be tested with `valgrind`.
- The performance of your code will be tested against a `std::string`. Your code must be faster than the `std::string` implementation; however, it is likely your code will be much faster if implemented correctly.
- Just running make will run the test, and then run the leak test afterward if it succeeds.

Below is an example output of a successful run on the CSIF.

```
mkdir -p ./obj
mkdir -p ./testbin
g++ ./src/teststr.cpp -o ./obj/teststr.o -c -Wall -std=c++14   -I ./include
gcc ./src/SafeString.c -o ./obj/SafeString.o -c -Wall  -I ./include
g++ ./obj/SafeString.o ./obj/teststr.o -o ./testbin/teststr -Wall -std=c++14
-lgtest_main -lgtest -lpthread
./testbin/teststr
Running main() from gtest_main.cc
[==========] Running 7 tests from 1 test case.
[----------] Global test environment set-up.
[----------] 7 tests from SafeString
[ RUN      ] SafeString.AllocateTest
[       OK ] SafeString.AllocateTest (0 ms)
[ RUN      ] SafeString.CreateTest
[       OK ] SafeString.CreateTest (0 ms)
[ RUN      ] SafeString.CopyTest
[       OK ] SafeString.CopyTest (0 ms)
[ RUN      ] SafeString.ConcatenateTest
[       OK ] SafeString.ConcatenateTest (0 ms)
[ RUN      ] SafeString.ResizeTest
[       OK ] SafeString.ResizeTest (0 ms)
[ RUN      ] SafeString.ErrorTest
[       OK ] SafeString.ErrorTest (0 ms)
[ RUN      ] SafeString.SpeedTest
[       OK ] SafeString.SpeedTest (301 ms)
[----------] 7 tests from SafeString (301 ms total)

[----------] Global test environment tear-down
[==========] 7 tests from 1 test case ran. (301 ms total)
[  PASSED  ] 7 tests.
valgrind --undef-value-errors=no --leak-check=full ./testbin/teststr
==30713== Memcheck, a memory error detector
==30713== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==30713== Using Valgrind-3.13.0 and LibVEX; rerun with -h for copyright info
```

```
==30713== Command: ./testbin/teststr
==30713==
Running main() from gtest_main.cc
[==========] Running 7 tests from 1 test case.
[----------] Global test environment set-up.
[----------] 7 tests from SafeString
[ RUN      ] SafeString.AllocateTest
[       OK ] SafeString.AllocateTest (7 ms)
[ RUN      ] SafeString.CreateTest
[       OK ] SafeString.CreateTest (3 ms)
[ RUN      ] SafeString.CopyTest
[       OK ] SafeString.CopyTest (3 ms)
[ RUN      ] SafeString.ConcatenateTest
[       OK ] SafeString.ConcatenateTest (5 ms)
[ RUN      ] SafeString.ResizeTest
[       OK ] SafeString.ResizeTest (3 ms)
[ RUN      ] SafeString.ErrorTest
[       OK ] SafeString.ErrorTest (5 ms)
[ RUN      ] SafeString.SpeedTest
[       OK ] SafeString.SpeedTest (6123 ms)
[----------] 7 tests from SafeString (6156 ms total)

[----------] Global test environment tear-down
[==========] 7 tests from 1 test case ran. (6178 ms total)
[  PASSED  ] 7 tests.
==30713==
==30713== HEAP SUMMARY:
==30713==     in use at exit: 0 bytes in 0 blocks
==30713==   total heap usage: 156,201 allocs, 156,201 frees, 9,126,412,579
bytes allocated
==30713==
==30713== All heap blocks were freed -- no leaks are possible
==30713==
==30713== For counts of detected and suppressed errors, rerun with: -v
==30713== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

NOTE: If you are testing on OS-X, there are known issues with `valgrind` displaying leaks even when some do not exist. Also on Mojave and newer, it may not be possible to get install `valgrind`, so you may want to develop on the CSIF.

You will submit only your `SafeString.c` file. You should avoid using existing source code as a primer that is currently available on the Internet. You **must** specify in the comments of your `SafeString.c` file any sources of code that you have viewed to help you complete this project. All class projects will be submitted to MOSS to determine if students have excessively collaborated. Excessive collaboration, or failure to list external code sources will result in the matter being referred to Student Judicial Affairs.