Secure Coding (CNT 4419)
Assignment I

Objectives: To become acquainted with a C safe-string library and use it to implement basic string operations.

Due Date: Sunday, October 30, 2022 at 11:59pm. No late submissions will be accepted.

Assignment Description
Complete this assignment by yourself.

While doing this assignment you will need to run a C compiler. One option is to use an online C compiler such as https://paiza.io/en/languages/online-c-compiler. If you use Paiza, you may create an account to save your code. Paiza has an Input tab for providing input to the program.

This assignment asks you to use the Better Strings Library. An example program (*Main.c*) using this library can be found at https://paiza.io/projects/OwRf7NzAYfmP5VDfpcMr6A. This example includes sample input in the Input tab. You may fork this code (that is, add a copy to your account) using the menu next to the Run button.

If you are not using Paiza, begin by downloading the *bstrlib.h* and *bstrlib.c* files from the repository at https://github.com/websnarf/bstrlib. Then copy-paste the code shown in *Main.c* at the Paiza link above into your own *Main.c* file.

As part of the Better Strings Library, you will be using the `bgets`, `bdestroy`, `bconcat`, and `binstr` functions, which are declared and implemented in the *bstrlib.h* and *bstrlib.c* files. These functions manipulate values of type `bstring`, a "better" string type (i.e., safer than the normal C-string type). Values of type `bstring` have two fields important for this assignment: `data` and `slen`. Field `data` refers to the underlying string (of type `unsigned char *`), and field `slen` refers to the string's length (of type `int`). Your program can access these fields directly.

Below is documentation for the library functions you will need to use. This documentation appears in the *bstrlib.txt* file in the GitHub repository.

> **bgets**
> extern bstring bgets (bNgetc getcPtr, void * parm, char terminator);
> typedef int (* bNgetc) (void * parm);
>
> Read a bstring from a stream. As many bytes as is necessary are read until the terminator is consumed or no more characters are available from the stream. If read from the stream, the terminator character will be appended to the end of the returned bstring. The getcPtr function must have the same semantics as the fgetc C library function (i.e., returning an integer whose value is negative when there are no more characters available, otherwise the value of the next available unsigned character from the stream.) The intention is that parm would contain the stream data context/state required (similar to the role of the FILE* I/O stream parameter of fgets.) If no characters are read, or there is some other detectable error, NULL is returned.

bgets will never call the getcPtr function more often than necessary to construct its output (including a single call, if required, to determine that the stream contains no more characters.)

Abstracting the character stream function and terminator character allows for different stream devices and string formats other than '\n' terminated lines in a file if desired (consider \032 terminated email messages, in a UNIX mailbox for example.)

For files, this function can be used analogously as fgets as follows:

fp = fopen (...);
if (fp) b = bgets ((bNgetc) fgetc, fp, '\n');

(Note that only one terminator character can be used, and that '\0' is not assumed to terminate the stream in addition to the terminator character. This is consistent with the semantics of fgets.)

[Hence,

```
bstring b = bgets ((bNgetc) fgetc, stdin, '\n');
```

reads a "better string" from stdin.]

bdestroy
extern int bdestroy (bstring b);

Deallocate the bstring passed. Passing NULL in as a parameter will have no effect. Note that both the header and the data portion of the bstring will be freed. No other bstring function which modifies one of its parameters will free or reallocate the header. Because of this, in general, bdestroy cannot be called on any declared struct tagbstring even if it is not write protected. A bstring which is write protected cannot be destroyed via the bdestroy call. Any attempt to do so will result in no action taken, and BSTR_ERR will be returned.

bconcat
extern int bconcat (bstring b0, const_bstring b1);

Concatenate the bstring b1 to the end of bstring b0. The value BSTR_OK is returned if the operation is successful, otherwise BSTR_ERR is returned.

binstr
extern int binstr (const_bstring s1, int pos, const_bstring s2);

Search for the bstring s2 in s1 starting at position pos and looking in a forward (increasing) direction. If it is found then it returns with the first position after pos where it is found, otherwise it returns BSTR_ERR. The algorithm used is brute force; O(m*n).

Use the `bstring` library to implement the following 3 tasks.

1. Read a string s, print s, and print the length of s.
2. Read strings s1 and s2, append s2 to s1, and print the result.
3. Read strings s1 and s2 and print a message indicating whether s2 is a substring of s1.

Most of Task 1 has already been implemented for you in the given *Main.c* file, but you still need to add a line of code to print the length of the input string. Note that the given code calls `rmNewLine` to remove any final newline character from the input string and calls `bdestroy` to deallocate `bstring` memory. Call `rmNewLine` and `bdestroy` as appropriate throughout this assignment.

Implement Task 2 using the `bconcat` function and Task 3 using the `binstr` function.

Submit to Canvas your completed *Main.c* file that performs the 3 required tasks.

Notes:

- For this assignment, only the *Main.c*, *bstrlib.h* and *bstrlib.c* files are needed. Your *Main.c* should include only the *bstrlib.h* and standard-I/O (*stdio.h*) header files.
- Only strings input by the user need to be handled by the Better Strings library. Static strings may be character arrays, as seen in the example *Main.c* you have been provided.

Sample Execution

Sample Input (note the space before Coding):

```
Input for Assignment 1
Secure
 Coding
Secure
cure
```

Sample Output:

```
The input string is: Input for Assignment 1
The length of this string is: 22
The concatenation of 'Secure' and ' Coding' is: Secure Coding
Finally, 'cure' is a substring of 'Secure'.
```

Changing the final line of input to Z would change the final line of output to:

Finally, 'Z' is not a substring of 'Secure'.

Optional Further Reading

The bstring library provides the following alternative means of accessing the data and slen fields.

> **char * bdata (bstring b);**
> Returns the char * data portion of the bstring b. If b is NULL, NULL is returned.
>
> **int blength (const_bstring b);**
> Returns the length of the bstring. If the bstring is NULL, the length returned is 0.

These accessors are documented in the *bstrlib.txt* file in the GitHub repository.

However, above this documentation, the following notice appears.

> The macros described below are shown in a prototype form indicating their intended usage. Note that the parameters passed to these macros will be referenced multiple times. As with all macros, programmer care is required to guard against unintended side effects.

Macros are another source of security vulnerabilities in C. Details can be found in Chapter 2 of *The CERT C Secure Coding Standard* by Robert C. Seacord. (Reading from this book is entirely optional and not required for this course.)