Project 1: Username Email and Password Verification

Due Oct 18, 2021 by 11:59pm **Points** 150 **Submitting** a file upload **File Types** c
Available Sep 20, 2021 at 12am - Dec 24, 2021 at 11:59pm

This assignment was locked Dec 24, 2021 at 11:59pm.

Project 1: Username, Email, and Password Verification

In this project, we will be verifying the correct formatting of a username, email address, and password for registration to a website.

Corrections and Additions

- There is a typo in one of the Error Messages. In ERROR_02_USER_LEN_INVALID "characters" is spelled "charcters". Please retain the spelling error. The autograder uses exact text matching and will look for the misspelled version.
- Note The length of strings does not include the '\0\ character. abcdefghabcdefghabcdefghabcdefgh is a valid username and has exactly 32 characters.
- The comment on line 98 in the starter file functions_template.c (//mike.wisc.edu) is incorrectly placed. This type of input should result in a ERROR_07:"Invalid character in name\n" to be printed, **not** ERROR_08.
- ERROR_06_EMAIL_NAME_LEN_INVALID implies that fewer than 32 (strictly <32) characters are permitted in the name of the email, however 32 should be inclusive (should permit names <=32 characters). Please retain the error. The autograder uses exact text matching and will look for this version.

Learning Goals

This project will help you gain familiarity working with C programming and practice using arrays and pointers.

Files

- functions_template.c : this is the file where you will write your code, rename this file to **functions.c**
- functions.h : header file containing the function prototypes and error messages
- verify.c : the driver for this project. The file that contains main.
- The first thing you should do is rename functions_template.c to functions.c
- You will only be turning in functions.c. Your code must work with the original functions.h and verify.c files.
- Project1UserTests.zip : Test Files

Specifications

This project simulates creating an online account for a website. The program asks a user to enter their username, email address, and password. Then it verifies the correct formatting of each of these pieces of data, and either reports success or reports the first error and exits.

You will write or complete five functions. The formatting rules for the username, email, and password are found in the functions.c file in the documentation block for each function.

The text of the error messages is found in the header file functions.h

If the user enters an invalid username, email address, or password the function should print the appropriate error message and return 0. We have written all the print statements for you. The grading script uses exact output text matching. If you change the error messages or print extra text your code will fail the tests in the grading script.

The code should test the username, email address, and password and only report the first error message. Do not report everything wrong with an item, only the first error. The error messages are listed in order for you in the functions.h file.

For example, the username must begin with a letter [A-Z, a-z], have a maximum of 32 characters, and may only contain letters, digits, or the underscore [A-Z, a-z, 0-9, _]. These conditions are tested in this order, so if the username is “CS354isThe_Best_Most_AwesomeClass!@#$%^&*(Ever)!!!”. The first test will pass (this should not generate any output), but the second test will fail because the username has 50 characters. When the second test fails, the output “Max 32 characters” is printed, the function returns 0. The “Invalid character in username” error message is never printed.

If the tests pass, then a success message is printed.

All messages printed on the screen have been written for you! We test your code using exact match output testing. If you change any of the messages, the tests will fail to match.

The only libraries that you may use have already been included in the provided files. Do not add additional libraries including string.h. While the standard libraries may be a good source of inspiration on how to approach some of the tasks in this assignment, ultimately all the string manipulation code you write must be your own. The goal of this project is to gain experience working with C programming C style arrays.

Hints and Definitions

1. When reading the string entered by the user in Get_User_Data(), the fgets function may read the newline character. Search the string and replace the newline with ‘\0’
2. Use these guidelines to break the email address into its components.
   1. Declare an at_pointer variable and search the string for the @ symbol.
   2. Declare an end_pointer and search the string for the ‘\0’ character.
   3. Email addresses have 4 parts. The name, the @ symbol, the domain, and the top-level domain. The top-level domain is the last 4 characters and must be “.com”, “.edu”, or “.net” for this project. The domain is all the characters between the @ symbol and the top-level domain.
   4. The domain may be broken into several subdomains each separated by ‘.’
   5. For example, doescher@cs.wisc.edu has the name “doescher”, the domain is “cs.wisc”, and consists of 2 subdomains “cs” and “wisc”. The top-level domain is “.edu”
   6. Note the domain does not include the top-level domain.
   7. The last character of the name is the character before the '@' symbol, or if the @ symbol is missing then the last character of the name becomes the last character of the domain.
   8. For example, in the incorrect email address “doescher.cs.wisc.edu”. We would test “doescher.cs.wisc” as the name and return the error “Invalid character in name”.
   9. If both the @ symbol and the top-level domain are missing, test the entire string as the name
   10. Hint: Test each subdomain independently

Comments

The comments for each of the five function contain information about what each function is expected to do. For further clarification, you should inspect the functions.h file to see the error messages may be displayed for a given function.

Compiling and running

Use the following commands to compile both code files and run your program.

```
gcc -g -o web_account_verification verify.c functions.c -Wall
```

```
./web_account_verification
```

- gcc is the name of the compiler
- -g turns on debugging symbols
- -o indicates the next thing will be the name of the output file
- web_account_verification is the name of the output file. Choose an output filename that is quite different from any of your source code filenames.
- The .c files are your source code files
- -Wall turns on all warnings.
- ./ indicates the current directory

Test the Code

We are providing a few files with both input data and the appropriate output message that you can use to verify your code is working. This is not the complete set of tests we will run when we grade your project. We strongly encourage you to write many more test files to check each of the error messages.

```
./web_account_verification < userTests/t1
```

The above line is an example of how to run the provided test named t1. The “<” symbol represents the input redirection operator when used in the Bash terminal on the CSL Machines. This will take care of entering the input, so you don’t have to type it in manually. You can either open this file in a text editor or use the bash command cat to read it. The first line below assumes you have unzipped the test files in a directory called userTests and will print the contents of the file to the screen.

```
cat userTests/o1
```

```
./web_account_verification < userTests/t1 > myo1 && diff myo1 userTests/o1
```

For even greater efficiency the second line will run the test named t1 and view the differences in output when compared to the actual output from the solution named o1. The “>” symbol redirects the output that would normally be printed to the screen to a file called myo1. The “&&” runs a second bash command – “diff” which compares two files. If the files match exactly then diff will produce no output.

Test your code by running it and entering both valid and invalid usernames, email addresses, and passwords. If you come up with any tricky or clever tests, please post them to Piazza under the project1 heading to share with other class members.

Strategy

Write your code in small pieces and test each line written by printing out a message. This technique is called scaffolding. This can be done using the debugging macro provided to you in the starter file or by inserting your own print statements where needed. In either case, ensure that no debugging messages are displayed after you verify the success of your code.

This project is based on a real-world web site registration. The order the error messages are displayed in do not represent the easiest order to write the tests in. For example, to test the name part of the email address you need to know where the @ symbol is; however, a missing @ symbol is the fifth error message.

Use the order of the error messages to approach the problem. The specification in the comments describes the complete rules but may not reflect the order the error messages should be tested in.

There is a lot of repetition in the required tasks to verify each of the four pieces. Put the redundant code in helper functions, so you only have to write it correctly once (or correct it in one place). For example, all three data items require length verification. Write one helper function to test the length of the strings.

Turn in

Test your code on a CSL machine! Upload your functions.c file to Canvas. Be sure to double-check the file in Canvas to confirm that you haven’t uploaded the original template.

References

From CSAPP:

- Section 2.1.4 Representing Strings

From the C Programming Language:

- Section 1.9 Character Arrays

While these are just a few sections that may be useful to read to complete the assignment, much of the content in the first few chapters of both textbooks provide good background knowledge to help you become a successful C programmer. Reading the complete chapters that these sections have been drawn from is strongly encouraged.

Project 1

Criteria	Ratings		Pts
Compiles Successfully compiles without any warnings or errors.	**4.5 pts** **Full Marks** Successful compilation with no errors and 0+ warnings.	**0 pts** **No Marks** Cannot compile due to errors in the program.	4.5 pts
Timeout Program completes within a reasonable amount of time.	**4.5 pts** **Full Marks**	**0 pts** **No Marks** An program error has caused your program to hang indefinitely.	4.5 pts
Libraries Program only uses approved C libraries.	**4.5 pts** **Full Marks**	**0 pts** **No Marks** Any library beyond stdlib is disallowed for use in this assignment.	4.5 pts
Basic Username Tests	**20 pts** **Full Marks**	**0 pts** **No Marks**	20 pts
Difficult Username Tests	**1.5 pts** **Full Marks**	**0 pts** **No Marks**	1.5 pts
Basic Email Tests	**80 pts** **Full Marks**	**0 pts** **No Marks**	80 pts
Difficult Email Tests	**2.5 pts** **Full Marks**	**0 pts** **No Marks**	2.5 pts
Basic Password Tests	**32 pts** **Full Marks**	**0 pts** **No Marks**	32 pts
Difficult Password Tests	**0.5 pts** **Full Marks**	**0 pts** **No Marks**	0.5 pts
			Total Points: 150