Project 1
CS221: C and Systems Programming – Fall 2019

Deadline: October 21, 2019 at 11:59pm

Part I. Password Strength Meter

Weak passwords are common source of e-mail and social website accounts hacks. To alleviate this problem, programs, often, when the user chooses a new password, determine if it is a legitimately strong password. You are to write a function that validates the password strength. The restrictions that we define as our defined level of strength of passwords is the following: the password must be at least eight characters long, and must include at least one uppercase, one lowercase letter, and one digit. The password cannot contain any other characters than letters and numbers. In addition, it must contain at least one string (consecutive letters) of at least four letters long (uppercase letters, lowecase letters, or combination of both). Furthermore, the password should not contain the username (usernames in our login program are not case-sensitive). Remember, failing to write this piece of code in the program correctly may result in accepting weak and vulnerable passwords. Your code helps rejecting such passwords.

```
bool isStrongPassword(const char* username, const char* password) {
    // TODO: your code goes here.
}
```

```
Enter username: vahab
Enter new password: 1234
Your password is week. Try again!

Enter username: vahab
Enter new password: hello
Your password is week. Try again!

Enter username: vahab
Enter new password: 123vaHaBk789
Your password is week. Try again!

Enter username: vahab
Enter new password: meAceCS221
Strong password!
```

Part II. Default Password Generator

In this part, you will write a C function `generateDefaultPassword(char* default_password, const char* username)` that generates a default password randomly, which will be sent to the user to login. The users may wish to change the default password to a password of their own, which should pass the password strength meter you implemented in Part I. The default password generated by this function will be stored in a char array that the `default_password` is pointing to. Remember, the caller of this function (aka `main()`) must declare an array before passing it to this function, in order to avoid segmentation fault issues.

The default password must be created randomly (every character should be chosen randomly and independently from all allowed characters) and must have all the requirements of a strong password the user selects (Part I) except the following:

The default password **must not** be longer than 15 characters.
The default password **may or may not** satisfy the four consecutive letters requirement.

Implement a new function called `isStrongDefaultPassword(const char* username, const char* password)` which checks if the default password is strong based on the new constraints. This function is almost identical to your `isStrongPassword()` function in Part I, but needs some modifications based on the default password requirements. In a loop, your program should then keep creating random strings and check if all the password

requirements for the default password are met. Once a compliant password is generated, the loop stops and your program displays the generated password in the following form:

```
Generating a default password...
Generated default password:  x1B4fxH81I02
```

```
void generateDefaultPassword(char* default_password, const char* username)
{
    //TODO: your code goes here.
}
```

While you are limiting the length of your random password to 15 maximum length, remember that the size of your random password must be random as well.

For both parts, you can write as many helper functions of your own as you like, but as for functions not written by you, you may only use `strlen`, `strcmp`, `strcpy`, and `strcat` from `string.h`, and any functions defined in `stdbool.h`, `stdio.h`, `ctype.h`, `time.h`, and `stdlib.h`, only. Your three functions must do exactly as specified in this project. You must include all your code relevant to these three functions inside the functions itself. Your code must compile and run correctly on the Linux lab machines. If we cannot compile your code on the lab machines, you will receive no credit. A small portion of your project grade (5%) is reserved for good coding style and adequate comments. While good coding style includes many elements, in this class we only grade your projects on suitable variable name choice and indentation.

Submission Guidelines:
Prior to the deadline, upload one zip file containing only two files: your C source code, and status.txt, to Canvas. The ZIP file name must be in the following format: LastName_FirstName_StudentID_proj1.zip For instance if my student ID is 123456789 and I am submitting my solution for project 1, then I am going to compress `status.txt` and `default_password_generator.c`, and rename the zip file to: Pournaghshband_Vahab_123456789_proj1.zip Do not submit/include any other file such as the executable file. A good sanity check is to check your zip file for corruption by extracting (unzipping) it and testing whether it did compress it successfully. If we cannot unzip your submission, you will receive no credit.

```
Sample status.txt file:
Vahab Pournaghshband - Project 1 The program works as required. It compiles/runs and the
output matches the correct format to the letter. However, the style and formatting is incorrect
because I didn't include any comments.
```