Project Name: Password Validation

Student Name: Tran Hong Nghiep

Due Date: August 9th, 2023

I. Problem Definition

Write a program to validate user passwords based on certain rules. The program should prompt the user to enter a password and check if it satisfies the following conditions:

+ The password must be at least 8 characters long and not exceed 20 characters.
+ The password must contain at least one uppercase letter, one lowercase letter, one digit, and one special character (e.g., @, #, $, etc.).
+ The password should not contain spaces or any other characters outside the allowed set.
+ The program should use strings to store the password and functions to validate it. It should provide appropriate error messages for passwords that fail to meet the criteria.

As expected, users must follow the syntax to achieve all criteria in sequence. In logic, users should consider the password length before checking for typing errors in the password. Besides, users might not be aware of their typing errors or even follow the rules given by the programs.

By habit, users will instead input passwords based on memorable patterns for easier access, such as their phone number or their names with their year of birth. Users might never mistype any invalid keyword, but they will never improve password strength with higher character variants. Depending on the password difficulty, user passwords are expected to cover only alphabetic characters (either uppercase or lowercase) or digits, and the password length might not even surpass 8 characters. Using these guessable patterns, their password strength is unqualified and sooner to being hacked.

Hence, preventing hackers from stealing passwords is always an urgent concern for both programmers and users. Weak defense in password storage systems can occur due to missing security definitions in the syntax or user carelessness about password safety. In a scenario, programmers allow user passwords to be “free-will” (could be a few characters or even empty) and are not bothered about how users should manage their passwords. For that reason, instead of providing personal options for users, they blame programmers for the weak protection of the secure system, despite their low responsibility for creating their passwords.

Due to this conflict, programmers must prioritize syntax criteria to keep track of users following rules instead of expecting the user's password to be well-defined. programmers require testing every possible variant of each character based on the user password pattern before asking users to type it in specific lengths. Programmers should also list any possible password patterns from users that do not match the requirements to minimize unpredictable input errors from users. With this strategy, users must

Project Name: Password Validation
Student Name: Tran Hong Nghiep
Due Date: August 9th, 2023
consider that their password must be filled, containing valid characters defined by the rules of the program, before determining the desired length.

II. Expectations on User Password

1. Character Limitation

Before checking password length, programmers must list all characters required in this program. These characters range from 33 to 126 in the ASCII table. Programmers can search detailed lists of character indexes at lookuptables.com (Link: https://www.lookuptables.com/text/ascii-table).

Noticeably, the index range for special characters does not follow in sequence order. Programmers should include function libraries related to string-type variables such as <cctype> to search and sort character types.

Character Types	Decimal Index Range
Alphabetic characters	Uppercase: 65 - 90 Lowercase: 97 - 122
Digits	48 - 57
Special characters (aka Punctuations)	Group 1: 33 - 47 Group 2: 58 - 64 Group 3: 91 - 96 Group 4: 123 - 126

2. Password Syntax

Next, the password should be customizable but contains at least 1 for each character type to pass the second requirement. In basic, users will fail to set up a password if their password does not include all these character types. Some patterns below will result in dissatisfaction with the criteria.

Pattern Type	Example
Numerical Pattern	096753428 98754321 1111111
Alphabetic Pattern	NguyenVanA nguyenvana NGUYENVANA
Mixed Numer and Alphabet	alo12345 h3LL0nH1

Project Name: Password Validation
Student Name: Tran Hong Nghiep
Due Date: August 9th, 2023

	CODE0365
Adding “Space”	iD #0909 In some cases, if users decide to fill a string with “Space” only, that case will be counted as an invalid-character error instead of an empty-string error.
Punctuation-Only	!@#$%^

Last, this program should display instructions to create a password and warning messages when users make input errors. Besides, warning messages should be followed in order. The most important note is that users will only receive one of these messages corresponding to the specific error they are dealing with, since users have read the instructions. The process will be complete when none of no warning messages are displayed. The guideline below illustrates the flow of the password storage system.

Display instructions (1 Time Only: Assume users understand the instructions);
Ask users to prompt a new password;
Check syntax step by step:

1. If the new password contains none of the invalid characters => Move to Step 2
   If Step 1 is failed, then ask users to input again
2. If the new password includes uppercase letters => Move to Step 3
   If Step 2 is failed, then ask users to add uppercases
3. If the new password includes lowercase letters => Move to Step 4
   If Step 3 is failed, then ask users to add lowercases
4. If the new password includes digits => Move to Step 5
   If Step 4 is failed, then ask users to add digits
5. If the new password includes punctuation => Move to Step 6
   If Step 5 is failed, then ask users to add punctuation
6. If length is between 8 to 20 characters => Store the Password

If one of the requirements cannot be passed => Redo the Entire Process

III. Input Declarations

1. Global Variables

Variable Name	**Usage**
const int charLengthMin = 8 and charLengthMax = 20	Set up the limitation of password length in the range from 8 to 20 characters. It can be adjusted before the program execution.

Project Name: Password Validation
Student Name: Tran Hong Nghiep
Due Date: August 9th, 2023

int invalidChars	Count invalid characters in newPassword, including “Space”. By default, user input records 0 errors.
int missCharTypes	Decrease if the input includes at least 1 type of valid character. By default, user input misses 4 types in the total.
string newPassword	User Input will be stored in newPassword after the syntax-checking process. By default, newPassword is an empty string and unassigned to any value.

2. Functions for Password Validations
   A. string addInput()

This function will receive a password from user input to test if it follows the syntax. By default, this function will return the value of the user's password if the length of newPassword is greater than 8 characters or lower than 20 characters. This function includes 1 sub-function to check invalid and missing character types. In addition, the length of the password will always be assigned 0 until the requirements in the sub-function are satisfied.

Variable Name	**Usage**
string newInput	newInput assigns the value of newPassword to test syntax in the sub-function and later extracts the length of newInput. By default, the length of newInput will only record 0 until invalidChars and missCharTypes = 0.
int limitMin int limitMax int charsLength	LimitMin receives the value of charLengthMin. LimitMax receives the value of charLengthMax. charsLength will count the length of newInput. By default, this function will not return newInput when charLength = 0 until charLength is in the range from limitMin to limitMax.

   B. void checkInputSyntax()

This function will test if the password misses any valid character types or includes any invalid characters. By default, the function addInput() can only extract the actual length of the password until checkInputSyntax() checks if the password has all valid character types and no invalid characters are counted.

Project Name: Password Validation
Student Name: Tran Hong Nghiep
Due Date: August 9th, 2023

<table>
<tr><th>Variable Name</th><th>Usage</th></tr>
<tr><td>string newStr</td><td>Test input from newInput if newInput includes all valid character types and excludes invalid characters. If not, charLength will always be 0.</td></tr>
<tr><td>char randomChar</td><td>randomChar will scan each character in newStr. If randomChar is an invalid character, invalidChars will increase by 1 value.</td></tr>
<tr><td>int includeUpper
int includeLower
int includeDigit
int includePunct</td><td>If any of these character types are included at the first scan while randomChar is scanning, then that type remains the value 1 throughout the loop. Ex: newStr = “@3cD”
<table>
<tr><th>Loop</th><th>randomChar</th><th>include Digit</th><th>include Upper</th><th>include Lower</th><th>include Punct</th></tr>
<tr><td>1</td><td>‘@’</td><td>0</td><td>0</td><td>0</td><td>1</td></tr>
<tr><td>2</td><td>‘3’</td><td>1</td><td>0</td><td>0</td><td>1</td></tr>
<tr><td>3</td><td>‘c’</td><td>1</td><td>0</td><td>1</td><td>1</td></tr>
<tr><td>4</td><td>‘D’</td><td>1</td><td>1</td><td>1</td><td>1</td></tr>
</table>
If all these types are included, missCharTypes should be 0.</td></tr>
</table>

IV. Pseudocode

1. Main Flow of The Program

Declare const int charLengthMin = 8 and charLengthMax = 20;
Declare int invalidChars and missCharTypes;
Declare string newPassword;
Create a function string addInput(string newInput, int limitMin, int limitMax);
Create a function void checkInputSyntax(string newStr)
Add function checkInputSyntax() inside function addInput();
Display instructions of the program with requirements;
Display 1 valid example of a password.
Add newPassword, charLengthMin, and charLengthMax in parameters of the function addInput() in sequence order;
Assign the value in the function addInput() to newPassword;

2. string addInput(string newInput, int limitMin, int limitMax)

Declare int charLength;
Do these tasks below:

Project Name: Password Validation

Student Name: Tran Hong Nghiep

Due Date: August 9th, 2023

Ask users for new newInput;
Add newInput in the function checkInputSyntax();
If invalidChars and missCharTypes > 0, then assign charLength = 0;
Else, assign charLength = Actual length of newInput and check:
If charLength < limitMin, display a warning message;
Else if charLength > limitMax, display warning message;
Else, display a confirmation that the password is complete;
End if-else statements;
End loop until charLength >= limitMin or charLength <= limitMax;
Return value of newInput;

3. **checkInputSyntax(string newStr)**

Assign invalidChars = 0, missCharTypes = 4;
Declare int includeDigit = 0 and includeUpper = 0;
Declare includeLower = 0 and includePunct = 0;
Declare and check each char randomChar in newStr:
If randomChar < '!' or randomChar > '~', then invalidChars Increase +1;
Assign includeUpper = 1 if randomChar is uppercase or includeUpper > 0
Or else includeUpper = 0;
Assign includeLower = 1 if randomChar is lowercase or includeLower > 0
Or else includeLower = 0;
Assign includeDigit = 1 if randomChar is a digit or includeDigit > 0
Or else includeLower = 0;
Assign includePunct = 1 if randomChar is punctuation or includePunct > 0
Or else includePunct = 0;
End loop when all randomChar is checked;
Sum includeUpper, includeLower, includeDigit, includePunct;
Subtract missCharTypes from the previous sum and assign back to missCharTypes;
If invalidChars > 0, then display invalidChars with a warning message;
Else if missCharTypes > 0, then:
If includeUpper < 1, then ask for uppercases;
If includeLower < 1, then ask for lowercases;
If includeDigit < 1, then ask for digits;
If includePunct < 1, then ask for punctuations;
End if-else statements;

Project Name: Password Validation
Student Name: Tran Hong Nghiep
Due Date: August 9th, 2023

V. Exception Handling

The table below shows the possibility of input characters to create a password.

Invalid?	Uppercase?	Lowercase?	Digits?	Punctuation?	Length	Valid Status
					0	No
x					1	No
x					8	No
x	x	x	x	x	5	No
		x	x	x	4	No
	x	x	x		12	No
x	x	x	x	x	9	No
	x	x	x	x	4	No
	x	x	x	x	21	No
	x	x	x	x	10	Yes

Despite the high definition of the syntax, it should be noted that the input range cannot cover and deal with all unrecognizable inputs such as extended ASCII characters (Example: Ç, Æ, ¼, Σ, etc.) or shortcut keys such as copy, cut, and paste. By logic, these characters are invalid to create passwords and technically cause programs to end the program execution or display incorrect outputs. Therefore, the program should display at least a confirmed message to mark the password setup is complete to prevent these unpreventable errors.