Decisions in Visual Basic II

Topics

- if/else statements
- testing the "randomness" of random numbers
- introduction to concept of string patterns
- choosing appropriate numerical data types
- simple vs compound conditions

Password crack calculator

Disclaimer. The program that you develop for this exercise should **NOT** be considered a measure of the strength of a particular password. An obvious example would be the password '*P@ssw0rd*', which, according to the program, would take 147.53338 centuries to crack, but in practice would be among the first passwords checked by any cracker worth their salt.

3. Use **File Explorer** to navigate to your lab folder. Copy and paste the folder `PasswordCracking_Part2`, and then right-click on the copy to rename it as `PasswordCracking_Part3`. Launch VS Express and use it to open the project `PasswordCracking_Part3`.

   Now, let's address the error caused by inputting the password `"aA1@@@"`. Hopefully, you discovered that the correct search space for this password is 100,343,116,692. Unfortunately, this falls outside of the range of values that can be represented using the `Integer` data type, which is -2,147,483,648 through 2,147,483,647. Fix your project so that it handles passwords whose search space is larger than 2,147,116,692 by changing the declaration of your `searchSpace` variable to a more appropriate data type.

Your project should behave identically to the solution which can be run by double-clicking the file `PasswordCracking_Part3.exe` found in the `Executables` folder inside of your lab folder.

When done, save and close project `PasswordCracking_Part3`.

Checkpoint 5 (80/100): A successful `PasswordCracking_Part3` project:

- ☐ produces correct output for all test cases in Test Table 2
- ☐ must also have successfully completed Checkpoint 2

4. Use **File Explorer** to navigate to your lab folder. Copy and paste the folder `PasswordCracking_Part3`, and then right-click on the copy to rename it as `PasswordCracking_Part4`. Launch VS Express and use it to open the project `PasswordCracking_Part4`.

   Enhance your project so that along with all previous information it now also calculates and outputs the total number of seconds required to check the entire search space. The number of checks per second that the computer is able to do is a value that is input by the user. Be sure to include the units (seconds) in your output statement.

   Here are some benchmarks of what computers are capable of. A computer guessing a password online using a web app hitting a target site is capable of approximately one thousand guesses per second. A high-powered server guessing a password offline is capable of approximately one billion guesses per second. And a massively parallel multiprocessing clusters or grid guessing a password offline is capable of approximately one trillion guesses per second.

Your project should behave identically to the solution which can be run by double-clicking the file `PasswordCracking_Part4.exe` found in the `Executables` folder inside of your lab folder.

When done, save and close project `PasswordCracking_Part4`.

Checkpoint 6 (85/100): A successful `PasswordCracking_Part4` project:

- ☐ output the correct amount of time to check the search space
- ☐ must also have successfully completed Checkpoint 2

5. Use **File Explorer** to navigate to your lab folder. Copy and paste the folder PasswordCracking_Part4, and then right-click on the copy to rename it as PasswordCracking_Part5. Launch VS Express and use it to open the project PasswordCracking_Part5.

   One of the most important tasks when developing a piece of software that depends on user input for calculations is to *validate* the user's input. *Validating input* involves checking the value input by the user to make sure that it conforms to any requirements of your project.

   For our password crack calculator, this requirement is that the password string contains only values from the following character types:

   - lower-case letters [a-z],
   - upper-case letters [A-Z],
   - digits [0-9], and
   - special characters [^$/%@#].

   Enhance your project so that if the password input by the user includes any characters not listed in the types above, then an error message is displayed to the user and the project DOES NOT compute any of the values that it would otherwise. In other words, the button action returns immediately, without doing any computation. To display the error message, you should use a *MessageBox*.

   To find out if a String contains a character that does not belong to a set of characters, you can also use the Like operator. For example, to check if the String variable passwd contains any characters that are not vowels and display a MessageBox and end the button action if it does, you would use the following Visual Basic code:

```
If passwd Like "*[!aeiou]*" Then
    MessageBox.Show("The password " & passwd & " contains a non-vowel character")
    Return
End If
```

   The ! at the beginning of the set of characters in brackets [ ] indicates to look for characters not in the set aeiou.

Your project should behave identically to the solution which can be run by double-clicking the file PasswordCracking_Part5.exe found in the Executables folder inside of your lab folder.

When done, save and close project PasswordCracking_Part5.

Checkpoint 7 (90/100): A successful PasswordCracking_Part5 project:

- ☐ handles invalid input (passwords that contain characters not allowed), by output a meaningful error message
- ☐ must also have successfully completed Checkpoint 6

6. Use **File Explorer** to navigate to your lab folder. Copy and paste the folder `PasswordCracking_Part5`, and then right-click on the copy to rename it as `PasswordCracking_Part6`. Launch VS Express and use it to open the project `PasswordCracking_Part6`.

   To make your project more user friendly, it should present the time to search using the most appropriate unit of measurement from the following list:

   - seconds,
   - minutes,
   - hours,
   - days,
   - weeks,
   - years, or
   - centuries.

   For example, if the number of seconds to search the space is less than 60, it should use the unit *seconds*, however if it is greater than 60, but less than 60×60=3600, then it should use the unit *minutes* and so on, all the way up to *centuries*. To keep things consistent, we will define a year as 52 weeks. In Visual Basic this could look like the following:

```
If searchTime < 60 Then
    unit = "seconds"
ElseIf searchTime < 60*60 Then
    searchTime = searchTime / 60.0
    unit = "minutes"
End If
```

   It is important to note the update of the value `searchTime` in the `ElseIf` branch of the above statement. If the correct unit is minutes, then we must adjust `searchTime` appropriately from its original units of seconds, to the correct units, namely minutes. This is accomplished by dividing the `searchTime` by the number of seconds per minute, which is 60. No adjustment is necessary when `searchTime < 60` because the original units form `searchTime` is seconds.

Your project should behave identically to the solution which can be run by double-clicking the file `PasswordCracking_Part6.exe` found in the `Executables` folder inside of your lab folder.

When done, save and close project `PasswordCracking_Part6`.

Checkpoint 8 (100/100): A successful `PasswordCracking_Part6` project:

- ☐ correctly converts and outputs time units
- ☐ must also have successfully completed Checkpoint 7

To read more about this topic, please visit this site.

Submission Instructions

Your `M:\CS130\Labs\Lab04_YourLastName_YourFirstName` folder should contain your solutions to this and Tuesday's lab.

To submit your work, copy this folder and paste it to `N:/Handins/CS130/Lab04_Decisions` PRIOR to the end of lab today. Late submissions won't be accepted and will receive a grade of 0 instead.