# SYSTEM SECURITY (CS F321)
# SEM I 2024-2025
# ASSIGNMENT 1
# MAX MARKS: 20 (WEIGHTAGE: 10%)
# DEADLINE: 18th SEPTEMBER 2024, 2:00 AM
# (HARD DEADLINE, Late submission will incur penalty)

**NOTE: PLEASE READ THE ENTIRE DOCUMENT AND NOT JUST THE PROBLEM STATEMENT. THIS DOCUMENT CONTAINS IMPORTANT INFORMATION REGARDING SUBMISSION GUIDELINES, LATE SUBMISSION POLICY, PLAGIARISM POLICY AND DEMO GUIDELINES IN ADDITION TO THE PROBLEM STATEMENT.**

**ALLOWED PROGRAMMING LANGUAGE: C**

## PROBLEM STATEMENT:

In this Assignment, you are going to implement a **Proactive Password Checker Application**. There is a system, S, where **n** ($3 \leq \mathbf{n} \leq 10$) users are registered. Assume that the users are already registered to the system. As part of this assignment, you are not required to perform new user registration. The application will help the existing users to change their passwords with the help of the proactive password checker application. The data that was collected from each user at the time of registration included username, date of birth and password. This means that the first password of every user has already been created.

The password checker helps the registered users of the system to change their passwords by facilitating the creation of strong passwords. The passwords created need to follow certain requirements to be deemed as strong. If a created password does not satisfy all the requirements simultaneously (mentioned later), it is rejected by the checker. Assume that the first password that was created initially by the user is a strong password satisfying all the requirements.

The database for this system stores the following information for the registered users in a file named `masterfile.txt` – username, date of birth and name of the password file. For each user, `username` is of the form `name.surname` (all characters in lowercase), date of birth (the field name stored in `masterfile.txt` is `date_of_birth`) is stored in `dd-mm-yyyy` format (the separators used in the date should be – only) and password file (the field name stored in `masterfile.txt` is `password_file_name`) is the file which stores the last 10 passwords created by the corresponding user in descending order of date of creation (i.e., the most recent and current password is the first entry in the password file and the oldest password is the last entry in the password file). If there are less than 10 entries in the password file, it implies that the user has changed the password less than 9 times in total. If the password has been changed more than 9 times by a user, the oldest password(s) will not be present in the file. The name of the password file is created by taking the first letter from the name of the user, the first letter from the surname of the user and appending the string `pass` after these two characters. You can have a look at the following table for the names of the password files of the different users. You can assume that no two users have the same combination of first letters for names and surnames. Assume that `masterfile.txt` and the password files of the different users are already created and stored in the same directory where you are going to save the program(s) of the application. Note that for each user, only one password file should exist at any point of time. **Duplicate password files for the users are not allowed.**

The format of `masterfile.txt` is shown as follows. Note the underscores in the column names. Also, assume that consecutive words/strings are separated by a white space character only. Consider date to be a single string but

having – s (dashes) as separators. For eg., `ramesh.yadav` and 19-09-1985 are separated by a single white space character and 19-09-1985 and `rypass.txt` are separated by another whitespace character.

| username | date_of_birth | password_file_name |
|---|---|---|
| ramesh.yadav | 19-09-1985 | rypass.txt |
| puja.bedi | 23-08-1991 | pbpass.txt |
| arun.kumar | 01-11-1973 | akpass.txt |
| anjali.sharma | 30-01-1996 | aspass.txt |

The password file of each user has the following format. This is a sample password file of some user.

2px34sa19h.fS
lkA@!o90a$5p
m.M90a21gth*k
xCXtimPOT23!p
Abcd.1234.*S
9091@asdfOOP$

Note that 9091@asdfOOP$ is the oldest password and 2px34sa19h.fS is the most recent and current password. Also, this user has changed the password 5 times till now after setting the first password at the time of registration.

Assume that `masterfile.txt` and the password files will be provided to you in the formats mentioned above. This means that you will not be required to create these files through your implementation.

In order to use the password checker application, the user needs to first login using the `username` and the current password. The prompt for this will be (the next line will be displayed only after the user has entered the `username`)

```
Enter username:
Enter password:
```

The user needs to supply the two pieces of information one by one. You can assume that it is alright if the password appears in clear text in an unmasked form.

In case the user enters a wrong password, the user will be re-prompted to enter the password using the following message.

```
Wrong password! Enter password again:
```

If the user enters the password wrongly 3 times, the application will terminate with the message

```
Wrong password entered 3 times. Application exiting….
```

Note that in this case, no backoff time is used. This implies that when a user enters the wrong password, s/he will be re-prompted to enter the password again (provided that the user has not already entered the password wrongly 3 times) immediately without any delay.

The authentication mechanism prevents one user from modifying another user's password. Once the user has been successfully authenticated, s/he will be prompted to enter the new password using the following display message

```
Enter your new password (1st attempt):
```

The proactive password checker imposes the following constraints for the users for password creation. Hence, each password will be checked against these requirements.

**R1:** Each password should be at least 12 characters long. You can assume that the maximum permissible length of a password is 20 characters. However, the password checker will check only for the minimum length requirement of 12 characters.

**R2:** Each password should have at least one uppercase letter.

**R3:** Each password should have at least one lowercase letter.

**R4:** Each password should have at least one digit (0 – 9).

**R5:** Each password should have at least one special character out of ., @, !, #, $, %, ^, &, *, -, and _ (The first one is dot. The second last one is dash and the last one is underscore. Note that comma is not a special character to be considered during password change. Comma is used as a separator in the list of permissible special characters.). If any other special character is used, the password will be rejected.

**R6:** Each password should not have more than 4 characters consecutively same as the last 10 passwords stored in the password file. This requirement is case insensitive. So, for this requirement the permissible limit is of 4 characters. For eg., If one of the last 10 passwords was hellB0y@45623A, then the new password cannot be He@venhellb0T341 since hellb0 (they are lowercase ls, not ones after h and e) is common between the two passwords. In fact, He@venHEllB06 is also not acceptable because of the case insensitive nature of this feature.

**R7:** Each password should not contain the name and/or the surname portion of the username. This requirement is case insensitive. For this requirement, consider the entire name and the entire surname. For eg., If username is `ramesh.yadav`, then Ramesh.1095@8 or RAMESHa.1095@8 or Ryadav@#8971op are not valid passwords. However, R@mes$h.9997a is a valid password. Also, Ramdav0956@#! is a valid password.

**R8:** Each password should not contain more than 3 consecutive digits of the date of birth sequence of the corresponding user. However, 1 or 2 or 3 digits of the date of birth are allowed to be present in the password consecutively. For this requirement, consider the date of birth without the dash separators. For eg., If a user's date of birth is 17-11-2001, then gtahp@AQ$1711 or gtahp@AQ$2001 or gtahp@AQ$1120 are not valid passwords. However, gtahp@AQ$1171 or gtahp@AQ$0211 or gtahp@AQ$1100 or gtahp@AQ$1701 are valid passwords. The last one is valid since 1701 is not present consecutively in the date of birth 17-11-2001.

If a password does not satisfy all the above-mentioned requirements, it is rejected by the password checker and the user is prompted to create the password again. Note that all the requirements have to be satisfied simultaneously. The password checker informs the user which requirement(s) has/have not been satisfied (so as to help the user to make a better attempt). Note that the password can violate more than one requirement. However, if the first attempt of password creation fails, the system uses a backoff time of 8 seconds before asking the user to create the password again. The backoff time should be displayed as a timer counting down from 8 to 0 using a display message like

`First attempt failed.`
`Password does not …..`(list the violated requirements here explicitly and not just as **R1**, **R2**, etc. )**.**
`Wait for x seconds….`

where `x` will keep changing from 8 to 7, then to 6 all the way down to 0.

The following prompt messages will be displayed as follows in case the requirements are not satisfied.

- If **R1** is violated, the prompt message should be
  `Password does not contain a minimum of 12 characters.`
- If **R2** is violated, the prompt message should be
  `Password does not contain at least one uppercase letter.`
- If **R3** is violated, the prompt message should be
  `Password does not contain at least one lowercase letter.`

- If **R4** is violated, the prompt message should be
  `Password does not contain at least one digit.`
- If **R5** is violated, the prompt message should be
  `Password does not contain at least one of the allowed special characters.`
- If **R6** is violated, the prompt message should be
  `Password contains c characters consecutively similar to one of the past 10 passwords.`
  where `c` is the number of consecutively matching characters with one of the past 10 passwords and `c` > 4. Note that you need to mention the exact number of consecutively matching characters.
- If **R7** is violated, the prompt message should be
  `Password contains name portion of the username.` (if only name matches)
  or
  `Password contains surname portion of username.` (if only surname matches)
  or
  `Password contains name and surname portions of username.` (if both name and surname matches)
- If **R8** is violated, the prompt message should be
  `Password contains d digits consecutively similar to the date of birth.`
  where `d` is the number of consecutively matching digits with the date of birth and `d` > 3. Note that you need to mention the exact number of consecutively matching digits.

So now, say in the first attempt, **R2**, **R3** and **R5** have been violated. The prompt message should be

```
First attempt failed.
Password does not contain at least one uppercase letter.
Password does not contain at least one lowercase letter.
Password does not contain at least one of the allowed special characters.
Wait for x seconds…. (x counts down from 8 to 0)
```

Once 8 seconds are over, the user will be prompted to enter the new password again with the help of the following message

`Enter your new password (2nd attempt):`

If the second attempt fails, the system uses a backoff time of 16 seconds and displays the list of the violated requirements for the second attempt. In this case, the prompt message will be

```
Second attempt failed.
Password does not contain …….  or   Password contains ….
Password does not contain …...  or   Password contains ….
.
.
Wait for x seconds…. (x counts down from 16 to 0)
```

After 16 seconds, the user will be asked to create the password again using the following prompt.

`Enter your new password (3rd attempt):`

After the third failed attempt, a backoff time of 32 seconds is used before re-prompting the user. Again, a message like the ones mentioned for 1st and 2nd failed attempts will be displayed to the user, mentioning the requirement(s) that was/were not satisfied, but after a backoff time of 32 seconds. This message will be of the form

```
Third attempt failed.
Password does not contain …….  or   Password contains ….
Password does not contain ……..  or   Password contains ….
.
.
Wait for x seconds…. (x counts down from 32 to 0)
```

The user is prompted for the fourth time using the following message.

```
Enter your new password (4th attempt):
```

If the fourth attempt also fails, the password checker terminates with a message

```
All 4 attempts failed. You need to try again later.
```

If the user is able to create the password successfully within the 4 attempts, then the following prompt is shown.

```
Password changed successfully.
```

If the password is created successfully, the password will be added to the corresponding user's password file as the first entry. If before this change of password, the user had already changed the password 9 or more times (9 because 1 is for the first password and 9 subsequent changes), then oldest password(s) will be overwritten. Say, before a change, the contents of the password file of a user were

password10
password9
password8
password7
password6
password5
password4
password3
password2
password1

such that password10 is the current password and password1 was the password created at the time of registration. Assume the strings password1 to password10 to be strings that are legitimate passwords as per the requirements. Now if the user changes the password to password11, then the contents of the password file will be

password11
password10
password9
password8
password7
password6
password5
password4
password3
password2

Thus, password1 is removed from the file and password11 is added to the file as the first entry. Now, if the next time the user changes the password to password12 and password12 matches password1 (which is no longer present in the password file) with respect to more than 4 consecutive characters, the password checker will accept it as a valid password and add it to the password file. However, password12 should not match password2 with respect to more than 4 consecutive characters with any of the passwords present in the file. This means that the password is added to the password file only after it passes all the requirements.

Note that you should not use any other programming language other than C. You are not required to create any GUI. The interface required for user interaction is already included in the problem statement. **There should be only one copy of the password file for each user in the system S. Assume that each line of the files ends with the new line character. You can use the sample files attached with the assignment document for testing your application.**

## SUBMISSION GUIDELINES:

- **All program(s) should be C programs.**
- **All codes should run on Ubuntu 22.04 or 24.04 systems. You can be asked to demo the application on either system variant.**
- Submissions are to be done using the following Google Form: https://docs.google.com/forms/d/e/1FAIpQLScohiPGAWY-Te3wLxJkuRpKR6F_27Jx6hVMii0mMm4q-5VJKg/viewform
- There should be only submission per group.
- Each group should submit a zipped file containing all the relevant C programs and a text file containing the correct names and IDs of all the group members. The names and IDs should be written in uppercase letters only. The zipped file should be named as GroupX_A1 (X stands for the group number). You will be notified of your group number in a few days.
- Your group composition has been frozen now. You cannot add or drop any group member now.

## LATE SUBMISSION POLICY:

- The Google form will accept submissions beyond 2 AM, 18th September 2024.
- ***SUBMITTING AFTER THE DEADLINE WILL INCUR A LATE PENALTY.***
- **For every 15 minutes of delay, there will be a late penalty of 0.5 marks. However, no distinction will be made within that 15-minute window. For eg., If a group submits between 2:01 AM and 2:15 AM (inclusive), then that group will lose 0.5 marks. If the submission is made at 2:10 AM, then also the penalty will be of 0.5 marks.**
- It is your discretion what you want to do – take a penalty and correct some last minute error or submit a partially correct implementation (which will lead to marks deduction).
- For calculating the late penalty, the date and timestamp of the submission via the Google form will be considered. No arguments will be entertained in this regard.

## PLAGIARISM POLICY:

- All submissions will be checked for plagiarism.
- Lifting code/code snippets from the Internet is plagiarism. Taking and submitting the code of another group(s) is also plagiarism. However, plagiarism does not imply discussions and exchange of thoughts and ideas (not code).

- All cases of plagiarism will result in being awarded a hefty penalty. Groups found guilty of plagiarism may be summarily awarded zero also.
- All groups found involved in plagiarism, directly or indirectly, will be penalized.
- The entire group will be penalized irrespective of the number of group members involved in code exchange and consequently plagiarism. So, each member should ensure proper group coordination.
- The course team will not be responsible for any kind of intellectual property theft. So, if anyone is lifting your code from your laptop, that is completely your responsibility. Please remember that it is not the duty of the course team to investigate cases of plagiarism and figure out who is guilty and who is innocent.
- **PLEASE BE CAREFUL ABOUT SHARING CODE AMONG YOUR GROUP MEMBERS VIA ANY ONLINE CODE REPOSITORIES. BE CAREFUL ABOUT THE PERMISSION LEVELS (LIKE PUBLIC OR PRIVATE). INTELLECTUAL PROPERTY THEFT MAY ALSO HAPPEN VIA PUBLICLY SHARED REPOSITORIES.**

**DEMO GUIDELINES:**

- The assignment also consists of a demo component to evaluate each student's effort and level of understanding of the implementation.
- You will be supplied with the `masterfile.txt` and the password files during the demo.
- **The demos will be conducted in either the I-block labs or D-block labs. Therefore, the codes will be tested on the lab machines.**
- No group will be allowed to give the demo on their own laptop.
- **The codes should run on Ubuntu 22.04 or 24.04.**
- All group members should be present during the demo.
- Any absent group member will be awarded zero.
- The demos will be conducted in person.
- The demos will not be rescheduled.
- Though this is a group assignment, each group member should have full knowledge of the complete implementation. During the demo, questions may be asked from any aspect of the assignment.
- **Demos will be conducted on 21st September and 22nd September 2024. Note that demos can be conducted on any one or both days. So, you need to be available on campus for both days. You need to book your demo slots as per the availability of your entire group.**
- The code submitted via the Google form will be used for the demo. No newer version of the code will be considered for the demo.
- Each group member will be evaluated based on overall understanding and effort. A group assignment does not imply that each and every member of a group will be awarded the same marks.
- Any form of heckling and/or bargaining for marks with the evaluators will not be tolerated during the demo.

**********