

## COS 135 Individual Assignment 7

Due: Monday 03/27/23 End of the day

Write C programs for following tasks and submit your source codes (you must submit .c files without compilation errors). Sample Program inputs are highlighted in yellow.

What to submit:

- Please submit a .zip file with source codes. Your programs must produce similar outputs as given if the same inputs are provided.

**[-5 pts each]** Source codes should be able to compile and execute without errors or warnings:

- **[-5 pts for each compiler error]** Source codes should be able to compile without any errors
- Always compile your source codes with -Wall (enable all warnings) switch to find all the compiler warnings and fix them (see example below).

```
$ gcc mycode.c -o myprogram -Wall
```

**[-10 pts each]** **Comments** are required in the following locations (in each C source code):

- **[-2 pts]** At the top of the source code, comment your name and insert a short program description
- **[-4 pts]** Comment the purpose of each variable.
- **[-4 pts]** Comment major sections of your code such as inputs, functions, and outputs.

Program Design: Your program is a professional document and must be neat and easy to read.

All programs should follow these specifications.

- Comments should be aligned and entered in a consistent fashion
- Blank lines should be added to aid readability
- Code within blocks should be indented
- Comments should not contain spelling mistakes
- Variable names should be meaningful

**(a) (40pts):** Write a C program to generate and save 200 random integers between 0 and 999 (including 0 and 999) in an array. **Your program should contain two functions** to find the lowest and highest numbers in the array. The main function should call these two functions to pass the array and each function returns the index of the lowest or highest value (if fails, return -1).

Sample code skeleton is provided below. Please modify the code accordingly to output the index and value of the lowest and highest numbers in the following format:

Lowest number 14 is at index 34  
Highest number 999 is at index 345

```
#include<stdio.h>
#include<stdlib.h>

/*
return the index of the lowest number in the array
e.g., if the lowest number is in 3rd position, returns 2
if not return -1
*/
int find_lowest(int numbers[]) {

    return -1;
}

/*
return the highest number in the array
e.g., if the highest number is in 1st position, returns 0
if not return -1
*/
int find_highest(int numbers[]) {

    return -1;
}

int main() {

    int numbers [200];

}
```

**(b) (60pts):**

[40 pts] Write a C program to validate a user's password. A user should be able to enter their preferred password via keyboard input. Your program checks for specific criteria given below and returns **Valid Password** or **Invalid Password**. If the user's preferred password is invalid, it also outputs all the reasons as sample output shows.

Write separate functions to check the following criteria of the preferred password (each function takes user entered password as a parameter, and returns **1** if it meets a certain criterion else returns **0**):

- A function to test the password is at least eight characters long
- A function to test the password has at least one uppercase letter and one lowercase letter
- A function to test the password has at least one digit
- A function to test the password has at least one of these four characters: ! @ # \$

Sample output #1:

```
Please enter your preferred password: UOM@orono4469
Valid password
```

Sample output #2:

```
Please enter your preferred password: Uom4469
Invalid password
Reason #1: your password should contain at least eight characters
Reason #2: your password should contain at least one of the four
special characters "! @ # $"
```

Sample output #3:

```
Please enter your preferred password: UOM
Invalid password
Reason #1: your password should contain at least eight characters
Reason #2: your password should contain at least one lowercase
character
Reason #3: your password should contain at least one of the four
special characters "! @ # $"
Reason #4: your password should contain at least one digit
```

[20 pts] Now, extend your code to encrypt a valid password. Once your program detects a valid password, apply below **encryption algorithm** and output the encrypted password. You may define new functions where necessary.

Encryption algorithm is based on corresponding ASCII values of different characters in the valid password. For each character in the password,

- add +1 if it is alphanumeric. For example, if the character is an uppercase letter, lower case letter, or a digit:  $\text{encrypted\_character} = (\text{character} + 1)$ ;
- add -1 if it is one of the four special characters. For example, if the character is one of these four characters ! @ # \$:  $\text{encrypted\_character} = (\text{character} - 1)$ ;

Examples:

**C0S135#umaine** should be encrypted as **DPT246"vnbjof**

**U0M@orono4469** should be encrypted as **VPN?pspop557:**