Department of Computer Science and Engineering
Class Test - 3, Section - 16

Subject : Programming and Data Structure (CS19003)

Date: 11th March 2022 Time: 9:00 AM to 11:00 AM Marks: 100

Instructions for :

- Give the name of the programs files as <Your roll>_<test number>_<question number>.c. For example, 21XX12345_T1_Q1.c for question 1 of test 1 of a student with roll 21XX12345. **Follow the file naming convention strictly**.
- Apart from the main .c file for each program, you should also upload one additional temporary .c file for each program (such as when you have finished half of the code). The naming for the temporary file should be in the format <Your roll>_<test number>_<question number>_temp.c. For example, 21XX12345_T1_Q1_temp.c **Make sure that your main code do not deviate much from its temporary code for each program**.
- You should upload the main .c file and the temporary .c file individually to the Moodle course web page once you finish answering each question. No need to zip the files.
- The **deadline** to upload the programs is 12:00 Noon. Beyond that, submission will be closed (No extensions will be granted).

Part-A [Compulsory]
30 Marks

1. Take input an integer number of 'n' digits and check whether the number is a **Narcissistic number** or not. A Narcissistic number is a number which is equal to the sum of its digits, each raised to the power of the number of digits of the original number. For example, 153 is Narcissistic since it is of 3 digits and $153 = 1^3 + 5^3 + 3^3$. Similarly 1634 is also Narcissistic as $1634 = 1^4 + 6^4 + 3^4 + 4^4$. You must use the following function,

   - **int isNarcissistic(int n)** - checks whether a number is Narcissistic or not. Returns 1 if Narcissistic, 0 otherwise.

   You must take the original number as input from main() and call the respective function and display the appropriate result in main(). You may assume that the input will always be a positive integer of commensurate number of digits.

   Example:

   Input:

   153

   Output:

   Number of digits = odd

   Number is a Narcissistic number

Input:

632

Output:

Number of digits = odd

Number is not a Narcissistic number

Part-B [attempt any one question]

70 Marks

2. You have used passwords in all of your online accounts. A good password is an alphanumeric string where the different constituent characters are randomly placed throughout the length of the string. We can use a structure data type with the following attributes to represent a password in a computer.

struct password:

- int Length: Length of the password in number of characters
- int Uppercase :Number of uppercase characters in the password
- int Lowercase : Number of lowercase characters in the password
- int Numbers : Number of numerical characters, 0-9
- int Special: Number of special ASCII characters
- int Size : Size in bytes of the password (one character is equal to 8 bits)
- int Strength: Denotes the strength of the password, defined as the number of characters between the first uppercase and the first next lowercase character of the password, both the characters inclusive. For example a password 'XVBghy7J' has a strength of 4 (number of characters from 'X' till 'g'), 'ghfVbn@' has a strength of 2 (starting from 'V' and ending at 'b'). In case a password has no next lowercase characters after the first uppercase character, then strength is 1 (for example 'ghjtJLI' or 'fb$76bhK') and in case the password has no uppercase character at all, the strength is zero (for example, 'fft67%uj')
- char Content[50] : The actual content of the password, with a maximum length of 50 characters

Declare an array of structure of type password to hold 'n' different passwords, 'n' taken as input from the user initially. Hence enter 'n' different alphanumeric passwords, each having arbitrary size and random characters, but maximum size of each password is 50 characters. For each of the 'n' passwords, compute the different password attributes as shown above to fill up the parameters of the structure variables within the array. You may assume that the password consists of only lowercase 'a-z', uppercase 'A-Z', numericals '0-9' and special ASCII characters. There should not be any space within the password. Hence for each unique pair of the passwords among all the 'n' passwords (i.e if there are three passwords, then consider the pairs (1-2), (1-3) and (2-3)), compute the similarity index of the password pair. Similarity index for a password pair is defined as the number of password attributes that are equal for both the passwords, starting from the length upto the strength attribute, except the last attribute (char content[]) and expressed as a percentage. Since there are 7 attributes, if all the 7 attributes match for both the passwords, the similarity index is 100%, if none of the attributes match the index is 0%, if only 2 attributes match the index is (2/7)*100 = 28.57% and likewise. Declare appropriate data types. You can use strlen() **ONLY** and no other in-built string functions. You may however declare and use user-defined functions as you feel necessary.

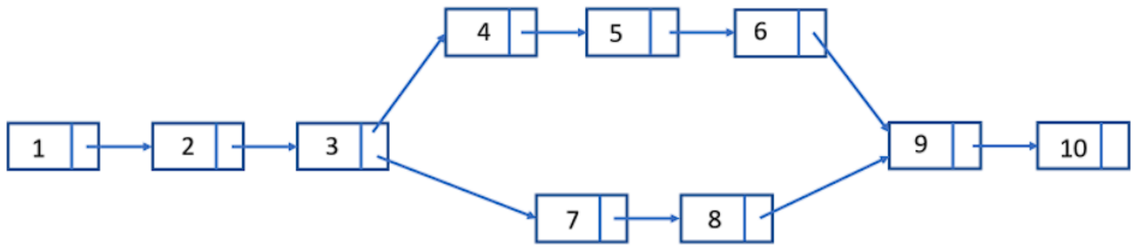
3. READ THE QUESTION CAREFULLY:

   Suppose you want to travel from source location ‘a’ to destination location ‘z’. So you start moving from location $a \to location\ b \to \cdots \to location\ x \to location\ z$. However, while traveling from location ‘a’ to ‘z’, there comes a point where the road diverges and proceeds via two different routes in a branching, which again converges after reaching a particular location. For example, consider the figure given below, location 1 is the source and location 10 is the destination. You can travel from location 1 to location 10 by any of the following two routes,

   $loc\,1 \to loc\,2 \to loc\,3 \to loc\,4 \to loc\,5 \to loc\,6 \to loc\,9 \to loc\,10$

   loc 1 $\to loc\,2 \to loc\,3 \to loc\,7 \to loc\,8 \to loc\,9 \to loc\,10$

   Here, location 3 is the divergence point for the branching and location 9 is the convergence point. Suppose now that you want to know the location of divergence and convergence of the road before traveling so that you can travel through the less congested road. To perform the above operation, firstly define a linked list node named ‘Transport‘ that contains one integer variable ‘Loc’ to denote the location identity and two pointer variables ‘Transport *Next_loc1’ and ‘Transport *Next_loc2’ that point to the next location that one can visit from that node towards the ultimate destination. If from a node there is only one next location that can be visited then that is pointed by *Next_loc1 and *Next_loc2 is NULL. If from a node two locations can be visited next, then the two locations are pointed to by the two pointer variables.



   Write a C program that takes the location information from the user and displays the divergence and convergence location. The user must first input the first route from source to destination and then the second route from source to destination. While taking the input routes, the linked list must be created accordingly. For the first route, create the linked list like a normal single chain linked list. You must use *Next_loc1 as the next pointing pointer and set *Next_loc2 as NULL, as you do not know about the second route, and hence any convergence or divergence yet. Then input the second route between source and destination location by location and first check whether the location is already covered in the first list. If yes, do nothing. If no, then at this point the path diverges, so create a new node for the second branch list and make the *Next_loc2 to point to this node from the diverging node (note that *Next_loc1 of the diverging node already points to the next node of the first branch). Carry on adding new nodes to this second route until again you find a common node with the first list that represents the convergence point. So make the *Next_loc1 pointer to point to the converging node and *Next_loc2 as NULL (reason already given above). Hence for all successive locations of the second route, it is the same as the first one, so do nothing. After taking input and creating the list, print the diverging and converging location. You may assume that the numerical integer ‘Loc’ is unique for every node and there is only one branching (i.e only one divergence and convergence point) between the source and destination, source and destination always being the same.

Example:

Input:

Enter the first route from source to the destination: 1 2 3 4 5 6 9 10

Enter the second route from source to the destination: 1 2 3 7 8 9 10

Output:

The divergence location is 3

The convergence location is 9

[**Note: 1.** You cannot check the convergence and divergence location at the time of taking the input. While taking the input, create the branched linked list and then check divergence and convergence locations by traversing the list and checking the respective pointers

2. Although you are not printing the entire list as output, **YOU MUST** implement this question with linked lists as defined. You **CANNOT** use arrays or simple scanning through the input sequences to identify the divergence and convergence]
