PDS-Lab (Section-17, Autumn 2020-21)
Class Test – 3 (1st March 2021, 2.30 – 5.30 PM) Marks = 100

Instructions:

- *a) Create a directory named as <rollno>_ct3, where <rollno> is your roll number.*
- *b) Give the name of the program as <p>.c where <p> implies the problems number, like 1.c 2.c 3.c etc. Store all the program under this class test in the directory <rollno>_ct3. Zip the entire directory <rollno>_ct3.*
- *c) You should upload your zipped file <rollno>_ct3.zip to the Moodle course web page latest by **5.30 PM** (without penalty). The **cutoff time** will be till **6.00 PM** with a penalty of **25%** on your secured marks (i.e., if you secured 80 marks, after penalty you will get 60 marks). Beyond 5.30 PM, the moodle system will not allow you to submit, as a result you will get zero.*
- ***d) Do not use library functions***
- *e) Penalty for plagiarism/copying: You will be awarded **0 (zero)** in the Test if you are involved in plagiarism/copying and an **additional 10 marks** will be deducted from overall PDS Lab marks.*
- *f) Keep your Camera **ON** (with **No virtual background**) throughout the Test. You should be always in front of the camera.*

1. Write a C program that will take a positive integer say n>=1 from the user, and will generate all the valid parenthesis sequences having n-left and n-right parenthesis. The program should print all the possible valid parenthesis sequences as shown below for the given n, and also display the total count of such valid sequences. **(20M)**

   For instance if n = 3, then all the possible valid parenthesis sequences are

   ```
   Parenthesis: ((()))
   Parenthesis: (()())
   Parenthesis: (())()
   Parenthesis: ()(())
   Parenthesis: ()()().
   ```

   Total number of valid sequences for n=3 : 5

2. Write a C program to create a file “stud_rec_file” and enter N (N to be specified by the user) student records from keyboard and write into the file. After, entering each record from the keyboard, immediately store the same into the file. Each student record consists of the following four fields: int roll_no, char name[20], int age, int marks[3]. After entering the N records, access the records one by one, and place them into 2 files (“pass_file” and “fail_file”) based on student’s performance. If the total marks is greater than or equal to 40%, then the student has passed, otherwise he/she has failed. Finally, read the contents of all three files, and display the same as shown below: **(20M)**

Example:
Number of student records N = 4

Contents of stud_rec_file:

Student record 1
Roll_no = 1
Name = XXXXXXX
Age = 13
Marks[3] = 47, 58, 83

Student record 2
Roll_no = 2
Name = yyyyyy
Age = 11
Marks[3] = 27, 38, 53

Student record 3
Roll_no = 3
Name = zzzz
Age = 12
Marks[3] = 74, 81, 13

Student record 4
Roll_no = 4
Name = wwwww
Age = 15
Marks[3] = 7, 12, 91

Contents of pass_file:
Student record 1
Roll_no = 1
Name = XXXXXXX
Age = 13
Marks[3] = 47, 58, 83

Student record 2
Roll_no = 3
Name = zzzz
Age = 12
Marks[3] = 74, 81, 13

Contents of fail_file:
Student record 1
Roll_no = 2
Name = yyyyyy
Age = 11
Marks[3] = 27, 38, 53

Student record 2
Roll_no = 4
Name = wwwww
Age = 15
Marks[3] = 7, 12, 91

3. Let us consider an m × n matrix (m and n are specified by the user) M, whose elements are binary bits (0 or 1) entered by the user through key board. Allocate the required memory space for the matrix M, during run-time using dynamic memory allocation. Write a C program to display all possible submatrices whose size is k × l (where k<=m and l<=n, k & l are specified by the user). The submatrix constituted by k rows and l columns should be contiguous. Find the Hamming distance between all possible pairs of sub-matrices, and determine the minimum and maximum Hamming distances.
Note: Hamming distance between two matrices is computed as the summation of the discrepancies in bit positions, when the matrices are completely superimposed over the other.

Hamming distance(S1, S2) = $\begin{matrix}1 & 0 & 0 & 1\\ 0 & 1 & 1 & 0\end{matrix} \quad \begin{matrix}0 & 0 & 1 & 0\\ 1 & 1 & 0 & 1\end{matrix} \quad = \quad 6$ **(35M)**

Example:
Size of a matrix M = 3 × 5

$$M = \begin{bmatrix}10010\\01101\\11101\end{bmatrix}$$

Size of the sub-matrix = 2 × 4

Possible submatrices are (s1, s2, s3, s4)

1001	0010	0110	1101
0110	1101	1110	1101

D(S1, S2) = 6
D(S1, S3) = 5
D(S1, S4) = 4
D(S2, S3) = 3
D(S2, S4) = 4
D(S3, S4) = 5

Maximum hamming distance = D(S1, S2) = 6
Minimum hamming distance = D(S2, S3) = 3

4. Write a C program that takes two sorted arrays (not necessarily of the same size) from the user and merge them to form a single sorted array by using a recursive function. User has to specify the sizes of two sorted arrays say m and n. Allocate the required memory space at run time using dynamic memory allocation for accommodating the two sorted arrays as well as merged sorted array, as per the user specification. The function should return the merged sorted array to main(). Finally, print the two given sorted arrays as well as the computed single merged sorted array as shown below: **(25M)**
   Example:
   Enter the sizes of two sorted arrays : m =3 & n = 3
   Enter the elements of first sorted array: 5 6 9
   Enter the elements of second sorted array: 3 7 8
   The elements of merged sorted array is: 3 5 6 7 8 9.