DEPARTMENT OF COMPUTER SCIENCE

FACULTY OF NATURAL SCIENCES, JAMIA MILLIA ISLAMIA, NEW DELHI-110025
TEL.: +91-11-26980014 (DIRECT), +91-11-26981717 EXTN 3450/ 3452

MCA (II-Sem) Sessional Examination, 2021

CSC27: Lab-IV (ADS)

Date: Sunday, June 13, 2021 Max. Marks: 25 Time:9:00 AM – 11:00 AM

Instructions:

(i) Write your Roll Number, Name at the top of the program as comment.
(ii) Attempt **ALL** questions.
(iii) Send your program through mail at **jahir.jmi@gmail.com**.
(iv) The program sends after the 11:00 AM may be not accepted.
(v) The students are advised to not share their program with other student. In case they have shared it, marks will be deducted.

Q1. A square matrix A of order n x n is called *upper left triangular matrix (ULTM)* if all of its elements below the right diagonal must be zero (nulls). For example matrix A given in figure 1 (a) is an ULTM of order 3 x 3. A square matrix A of order n x n is called *lower right triangular matrix (LRTM)* if all of its elements above the right diagonal must be zero (nulls). For example matrix B given in figure 1 (b) is an LRTM of order 3 x 3. **(10)**

$$\begin{bmatrix} a_{0,0} & a_{0,1} & a_{0,2} \\ a_{1,0} & a_{1,1} & 0 \\ a_{2,0} & 0 & 0 \end{bmatrix} \qquad \begin{bmatrix} 0 & 0 & b_{0,2} \\ 0 & b_{1,1} & b_{1,2} \\ b_{2,0} & b_{2,1} & b_{2,2} \end{bmatrix}$$

Fig 1 (a): ULTM – A **Fig 1 (b):** LRTM – B

Define *SquareMatrix* class with *a[n][n]*, and *n* are data members where *a[][]* is a 2D array stores the matrix and *n* is the order of square matrix; *SquareMatrix(n)* constructor function, *read()* and *print()* member functions where constructor function is used to create 2D array of size nxn and set the value of n data member and *read()* and *print()* member functions are used to read and print the square matrix. Thereafter, define the *ULTM* class with *a[]*, and *n* are data members where *a[]* is a 1D array stores the non-null (zero) elements of the *upper left triangular* matrix using row major order mapping function and *n* is the order of *upper left triangular* matrix; *ULTM(n)* constructor function, *map(i, j), read()* and *print()* member functions where constructor function is used to create 1D array of required size and set the value of n data member, *map(i, j)* function map the indices of non-null elements to index of 1D array, and *read()* and *print()* member functions are used to read and print the *upper left triangular* matrix. Then define *SquareMatrix product(ULTM A, ULTM B)* C/C++ function that compute the product of two *upper left triangular matrices* with minimum number of multiplication operations. Then show these operations in *main()* function.

OR

Define *SquareMatrix* class with *a[n][n]*, and *n* are data members where *a[][]* is a 2D array stores the matrix and *n* is the order of square matrix; *SquareMatrix(n)* constructor function, *read()* and *print()* member functions where constructor function is used to create 2D array of size nxn and set the value of n data member and *read()* and *print()* member functions are used to read and print the square matrix. Thereafter, define the *LRTM* class with *a[]*, and *n* are data members where *a[]* is a 1D array stores the non-null (zero) elements of the *lower right triangular* matrix using row major order mapping function and *n* is the order of *lower right triangular* matrix; *LRTM(n)* constructor function, *map(i, j), read()* and *print()* member functions where constructor function is used to create 1D array of required size and set the value of n data member, *map(i, j)* function map the indices of non-null elements to index of 1D array, and *read()* and *print()* member functions are used to read and print the *lower right triangular* matrix. Then define *SquareMatrix product(LRTM A, LRTM B)* C/C++ function that compute the product of two *lower right triangular matrices* with minimum number of multiplication operations. Then show these operations in *main()* function.

Q2. Define LinkedList class with *LinkedList()* constructor and *create(n)* and *display()* member functions. Thereafter, write an efficient C/C++ *partition(LinkedList lst)* **(using swapping of the nodes of the linked list)** that partition the nodes of the linked list such that all node having values less than equal to the value of the first node will be before the first node and nodes whose values are greater than value of the first node will be after the first element. Then show these operations in *main()* function. **(15)**