Programming Practice: Matrix Operations with Dynamic Memory Allocation

1. Let A be an m×n matrix and B be an n×p matrix. Matrix multiplication of A and B is matrix C=A×B defined as the following:

$$\begin{bmatrix} c_{0,0} & c_{0,1} & \cdots & c_{0,p-1} \\ c_{1,0} & c_{1,1} & \cdots & c_{1,p-1} \\ \vdots & \vdots & \ddots & \vdots \\ c_{m-1,0} & c_{m-1,1} & \cdots & c_{m-1,p-1} \end{bmatrix} = \begin{bmatrix} a_{0,0} & a_{0,1} & \cdots & a_{0,n-1} \\ a_{1,0} & a_{1,1} & \cdots & a_{1,n-1} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m-1,0} & a_{m-1,1} & \cdots & a_{m-1,n-1} \end{bmatrix} \times \begin{bmatrix} b_{0,0} & b_{1,0} & \cdots & b_{p-1,0} \\ b_{0,1} & b_{1,1} & \cdots & b_{p-1,1} \\ \vdots & \vdots & \ddots & \vdots \\ b_{0,n-1} & b_{1,n-1} & \cdots & b_{p-1,n-1} \end{bmatrix},$$

where $c_{i,j} = \sum_{k=0}^{n-1} a_{i,k} \times b_{k,j}$.

Element $c_{i,j}$ is the inner product of the i-th row of matrix A and the j-th column of B. Write a C program to input three integers m, n, and p of matrix size matrix A, B, and C. Use *dynamic memory allocation* to create *exact space* for matrices A, B, and C and randomly generate element $a_{i,j}$ and $b_{i,j}$ for matrices A and B. Then, compute matrix multiplication C=A×B. Output matrices A, B, and C. Program source code: **matrix_multiplication_dynamic.c**.

2. In some applications, matrices are sparse, instead of dense. A form of sparse matrix is triangular matrices, as the following figure:

$$A = \begin{bmatrix} a_{0,0} & 0 & \cdots & 0 & 0 \\ a_{1,0} & a_{1,1} & & 0 & 0 \\ \vdots & & \ddots & & \vdots \\ a_{r-2,0} & a_{r-2,1} & \cdots & a_{r-2,r-2} & 0 \\ a_{r-1,0} & a_{r-1,1} & & a_{r-1,r-2} & a_{r-1,r-1} \end{bmatrix},$$

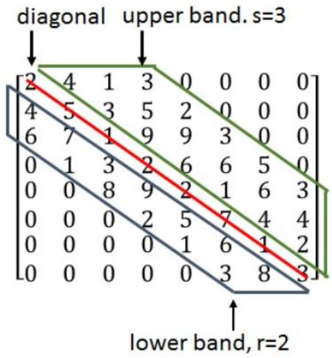
$$B = \begin{bmatrix} b_{0,0} & b_{0,1} & \cdots & b_{0,r-2} & b_{0,r-1} \\ 0 & b_{1,1} & & b_{1,r-2} & b_{1,r-1} \\ \vdots & & \ddots & & \vdots \\ 0 & 0 & \cdots & b_{r-2,r-2} & b_{r-2,r-1} \\ 0 & 0 & & 0 & b_{r-1,r-1} \end{bmatrix},$$

$$C = \begin{bmatrix} c_{0,0} & c_{0,1} & \cdots & c_{0,r-2} & c_{0,r-1} \\ c_{1,0} & c_{1,1} & & c_{1,r-2} & c_{1,r-1} \\ \vdots & & \ddots & & \vdots \\ c_{r-2,0} & c_{r-2,1} & \cdots & c_{r-2,r-2} & c_{r-2,r-1} \\ c_{r-1,0} & c_{r-1,1} & & c_{r-1,r-2} & c_{r-1,r-1} \end{bmatrix},$$

$$\forall\, 0 \le i,j < r, c_{i,j} = \sum_{k=0}^{\min(i,j)} a_{i,k} \times b_{k,j}.$$

A is a lower triangular matrix with all elements above the diagonal being 0's, B is an upper triangular matrix with all elements below the diagonal being 0's, and C=A×B. Write a C program to input an integer r of square matrix size for matrices A, B, and C. Use *dynamic memory allocation* to create *exact memory space* of matrix elements for triangular matrices A, B, and C and randomly generate element $a_{i,j}$, for i≥j, and $b_{i,j}$, for i≤j, for lower triangular matrix A and upper triangular matrix B, respectively. Then, compute matrix multiplication C=A×B. Output matrices A, B, and C; do not print the upper triangle elements for matrix A and lower triangle elements for matrix B. Program source code: **matrix_multiplication_triangular_dynamic.c**.

3. A kind of sparse matrix is banded matrix. If square matrix A is of size n×n, a lower band element of bandwidth r is the element $a_{i,j}$ such that 0<i-j≤r and an upper band element of bandwidth s is the element $a_{i,j}$ such that 0<j-i≤s. Only the elements on the diagonal, on the lower band, and on the upper band can be non-zero; all other elements are called off-band elements and they are all zeros. The following is an example of an 8×8 banded matrix with the lower bandwidth r of 2 and the upper bandwidth s of 3.



Let the size of square matrix A, B, and C be n×n. Also, let ra and sa be the lower and upper bandwidth of square matrix A, respectively, and rb and sb be the lower and upper bandwidth of square matrix B, respectively. If C=A×B, then the lower bandwidth of C is ra+rb and the upper bandwidth of C is sa+sb with the limit of upper bound n-1. The non-zero elements of $c_{i,j}$ is computed as the following formula:

$$\forall i,j: 0 \le i \le n-1 \wedge \max(0, i-ra-rb) \le j \le \min(n-1, i+sa+sb), c_{i,j} = \sum_{k=\max(0,\max(i-ra,j-sb))}^{\min(n-1,\min(i+sa,j+rb))} a_{i,k} \times b_{k,j}$$

Write a C program to input an integer n of square matrix size for matrices A, B, and C, and two pairs of integer, ra and sa as the lower and upper bandwidth of matrix A, and rb and sb as the lower and upper bandwidth of matrix B. Use *dynamic memory allocation* to create *exact memory space* of matrix elements for banded matrices A, B, and C and randomly generate non-zero $a_{i,j}$ and $b_{i,j}$ for matrices A and B. Then, compute matrix multiplication C=A×B. Output matrices A, B, and C; do not output the off-band elements form matrices A, B, and C. Program source code: **matrix_multiplication_banded_dynamic.c**.

4. Gaussian elimination is an algorithm for solving systems of linear equations. A variation of Gaussian elimination, known as LU-decomposition, is to factorize the n×n coefficient matrix A into two n×n triangular matrices, L and U such that A=L×U, as the following:

$$A = \begin{bmatrix} a_{0,0} & a_{0,1} & \cdots & \cdots & a_{0,n-1} \\ a_{1,0} & a_{1,1} & a_{1,2} & \cdots & a_{1,n-1} \\ \vdots & \vdots & \ddots & \cdots & \vdots \\ a_{n-2,0} & a_{n-2,1} & \cdots & a_{n-2,n-2} & a_{n-2,n-1} \\ a_{n-1,0} & a_{n-1,1} & \cdots & a_{n-1,n-2} & a_{n-1,n-1} \end{bmatrix},$$

$$L = \begin{bmatrix} 1 & 0 & \cdots & \cdots & 0 \\ l_{1,0} & 1 & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \cdots & 0 \\ l_{n-2,0} & l_{n-2,1} & \cdots & 1 & 0 \\ l_{n-1,0} & l_{n-1,1} & \cdots & l_{n-1,n-2} & 1 \end{bmatrix}, U = \begin{bmatrix} u_{0,0} & u_{0,1} & \cdots & \cdots & u_{0,n-1} \\ 0 & u_{1,1} & u_{1,2} & \cdots & u_{1,n-1} \\ \vdots & \vdots & \ddots & \cdots & \vdots \\ 0 & 0 & \cdots & u_{n-2,n-2} & u_{n-2,n-1} \\ 0 & 0 & \cdots & 0 & u_{n-1,n-1} \end{bmatrix}.$$

Let $A^{(k)}$ be the (n-k)×(n-k) sub-matrix of A after removing the first k rows and the first k columns, i.e.,

$$A^{(0)} = A,$$

$$A^{(k)} = \begin{bmatrix} a_{k,k} & a_{k,k+1} & \cdots & \cdots & a_{k,n-1} \\ a_{k+1,k} & a_{k+1,k+1} & a_{k+1,k+2} & \cdots & a_{k+1,n-1} \\ \vdots & \vdots & \ddots & \cdots & \vdots \\ a_{n-2,k} & a_{n-2,k+1} & \cdots & a_{n-2,n-2} & a_{n-2,n-1} \\ a_{n-1,k} & a_{n-1,k+1} & \cdots & a_{n-1,n-2} & a_{n-1,n-1} \end{bmatrix}, k \leq n-1.$$

Starting from $A^{(0)}$, matrices L and U are generated by computing, given $A^{(k)}$, $0 \leq k \leq n-1$, sub-matrices $A^{(k+1)}$ as the following steps:

1. Compute elements of the k-th row of matrix U: $u_{k,j}=a_{k,j}$, for $k \leq j \leq n-1$;
2. Compute elements of the k-th column of matrix L: $l_{i,k}=a_{i,k} / a_{k,k}$, for $k \leq i \leq n-1$; (note that, $l_{k,k}==1$)
3. Compute elements of sub-matrix $A^{(k+1)}$: $a_{i,j}=a_{i,j}-l_{i,k} \times u_{k,j}$, for $k<i, j \leq n-1$.

Write a C program to input an integer n of square matrix size for matrices A, L, and U. Use *dynamic memory allocation* to create *exact space* of matrix elements for matrices A, L, and U and randomly generate elements $a_{i,j}$, $0<a_{i,j} \leq 1$, for matrix A. Then, compute LU-decomposition $A=L \times U$ to generate matrices L and U. For the input matrix, keep a copy A1, and check whether $A1=L \times U$ to verify correctness of the program. Output matrices A, L, and U. Program source code: **lu_decomposition_dynamic.c**.

5. Suppose A is an $n \times n$ square matrix, i.e, the number of rows and the number of columns are n. When A is a 1×1 square matrix, the determinant of A, denoting |A|, is the value of the matrix element. When A is an $n \times n$ sqaure matrix, $n>1$, the determinant of A is defined recursively, expanding along the i-th row, below:

$$|A| = \sum_{j=0}^{n-1} (-1)^{i+j} a_{i,j} |cofactor(A, i, j)|.$$

where cofactor(A, i, j) is the $(n-1) \times (n-1)$ square matrix of A after removing the i-th row and j-th column elements. Write a C program to perform the following steps:

(1) Write a recursive function determinant() to compute the determinant of a square matrix. Use a global variable cnt to record the number of times function determinant() being called.
(2) Read a positive integer n between 1 and 12 as the number of rows and columns of square matrix A.
(3) Use *dynamic memory allocation* to to create *exact space* of matrix A. Then, use rand() to generate values for the elements of matrix A; each element is a floating point number between 0 and 1.
(4) Compute the determinant of square matrix A. In function determinant(), use *dynamic memory allocation* to to create *exact space* of cofactor(A, i, j). Each time when function determinant() is called, increment cnt by 1. Also, record the CPU time of computing determinant().
(5) Output square matrix A such that each element is of 2 digits after the decimal point and the determinant value of A with 6 digits after the decimal point.
(6) Output the number of times function determinant() being called and the CPU time using floating point format 4 digits after the decimal point.

Program source code: **determinant_dynamic.c**.