Programming Fundamentals (CS1002)

Fall 2022

|  |
| --- |

Develop a **Matrices System** by using all concepts learnt in the PF Course/Lab. This is a **group** project - there should be **2 students per group**. Both students in the group must be from the **same lab section** and must be self-selected.

# Project Description

In this project, you have to implement matrices system based on its order. The matrix must be a square matrix (i.e. rows and columns must be same). In the beginning, the program must ask the user to enter order (**N**) of matrix **A** and its elements. Then program should present the user a choice (1- 7) to perform specific tasks given below. For each task, make a separate function. You have to decide yourself the parameters/arguments and return type of each function. To terminate the program, user can enter 8.

1. To display a matrix
2. To check whether the entered matrix is symmetric matrix or not
3. To check whether the entered matrix is identity matrix or not
4. To determine the determinant of entered matrix
5. To determine the adjoint of entered matrix
6. To determine the inverse of entered matrix
7. To determine the solution of entered matrix (i.e. solving multiple linear equations simultaneously)

**Note:** User can find solution of multiple linear equations simultaneously through matrices either by Cramer’s Rule or by matrix inversion method. In both methods, user also have to enter the constant matrix **B** other than matrix **A.** The dimensions of matrix **B** should be **N x 1.** Following example explains how multiple linear equations are solved simultaneously through matrices either by Cramer’s Rule or by matrix inversion method.

**Example:**

To solve the above equations simultaneously through matrices, following matrices can be derived from above equations:

Let **‘D’** represents the determinant of matrix **A**:

**Finding the values of x,y and z using matrix inversion method:**

**Finding the values of x,y and z using Cramer’s Rule:**

If D ≠ 0, then

, ,

# Project Assessment:

In the course, the project will be evaluated under the CLO # 5.

|  | **Project Deliverables** | **LLO No[[1]](#footnote-0)** | **Marks** |
| --- | --- | --- | --- |
| 1. | Correct use of data types, input output statements and relational operators | 1 | 5 |
| 2. | Correct algorithm of the program:   * Data is stored in arrays correctly. * Sequence of function calls | 2 | 5 |
| 3. | Correct use of all functions:   * Argument and return type * Functionality | 3 | 15 |
| 5. | Absence of errors (syntax and logical) or warnings | 2 | 5 |
|  | **TOTAL MARKS** | **30** | |

NAME: MANAHIL FATIMA

CLASS:BSR-A

ROLL NO: 22I-2329

BREAKPOINT: I did the code all by myself with the permission of my instructor. i used pointers to implement my code as i find them easier to understand.

code with pointers

| //  // main.cpp  // final project  //  // Created by syeda manahil fatima on 09/12/2022.  //  #include <iostream>  #include<iomanip>  **using** **namespace** std;  **void** display(**int** \*arr, **int** c) // function definition  //pointers are used because columns are user defined  {  **for**(**int** i=0; i<c; i++)  { //function body  **for**(**int** j=0; j<c;j++)  {  cout<< \*(arr+i\*c+j) <<'\t'; //pointers use memory addresses.this logic is row\*column to print correct values  }  cout<<endl;  }  }  **void** symmetric(**int** \*arr,**int** c) //function to check if a matrix is symmetric or not  {  **int** count=0; //declaring and initializing a counter  **for**(**int** i=0; i<c;i++){  **for**(**int** j=0; j<c;j++){  **if**(\*(arr+i\*c+j)==\*(arr+j\*c+i)){ //checking transpose of the matrix. if the original matrix and transpose is same than it will be a symmetric matrix  count=count+1;  }  }  }  **if**(count==c\*c){ //condition is made to check if the counter is equal to the total number of elements of matrix. if equal than it means all values are verified  cout<<"it is a symmetric matrix"<<endl;  }  **else**  cout<<"not a symmetric matrix"<<endl;  }  **void** identity(**int** \*arr, **int** c)  { **int** count=0,counter=0;  **for**(**int** i=0;i<c;i++){  **if**(\*(arr+i\*c+i)==1){ //condition to check all values at main diagonal are equal to one  count++;  }  }  **for**(**int** i=0;i<c;i++){  **for**(**int** j=0; j<c; j++){  **if**(i!=j){ //to check if all values other than main diagonal are zero  **if**(\*(arr+i\*c+j)==0){  counter++;  }  }  }  }    **if**(count==c && counter==(c\*c)-c){ //condition to check if both upper conditions are true only than it is an identity matrix  cout<<"it is an identity matrix"<<endl;  }  **else**  cout<<"it is not an identity matrix"<<endl;  }  **double** determinant(**int** \*arr, **int** c)  {  **int** l,m,n,deter;  **if**(c==2){ //for a 2x2 matrix detrminant  deter=\*(arr+0\*c+0)\*(\*(arr+1\*c+1))-(\*(arr+0\*c+1))\*(\*(arr+1\*c+0));  }  **else** **if**(c==3) //now for 3x3 matrix  {  l=(\*(arr+1\*c+1)\*(\*(arr+2\*c+2)))-(\*(arr+2\*c+1)\*(\*(arr+1\*c+2))); //deleting first row and first column  m=(\*(arr+1\*c+0)\*(\*(arr+2\*c+2)))-(\*(arr+2\*c+0)\*(\*(arr+1\*c+2))); //deleting first row and second column  n=(\*(arr+1\*c+0)\*(\*(arr+2\*c+1)))-(\*(arr+2\*c+0)\*(\*(arr+1\*c+1))); //deleting first row and third column  deter=(l\*(\*(arr+0\*c+0)))-(m\*(\*(arr+0\*c+1)))+(n\*(\*(arr+0\*c+2)));  }  **return** deter; //it will return value stored in deter variable to main body  }  **void** adjoint(**int** \*arr, **int** c)  {**int** adj[c][c];    **if**(c==2) //for 2x2 MATRIX  {  adj[0][0]= \*(arr+1\*c+1); //to interchange values of diagonal  adj[1][1]=\*(arr+0\*c+0);  adj[0][1]=-(\*(arr+0\*c+1)); //multiplying non-diagonal entities with negative sign  adj[1][0]=-(\*(arr+1\*c+0));    **for**(**int** i=0; i<c; i++){  **for**(**int** j=0; j<c; j++){  cout<<adj[i][j]<<" "; //displaying adjoint matrix  }  cout<<endl;  }  }    **if**(c==3) //for 3x3 matrix  {  adj[0][0]=(\*(arr+1\*c+1)\*(\*(arr+2\*c+2)))-(\*(arr+2\*c+1)\*(\*(arr+1\*c+2))); //determinant after deleting 1st row and 1st column  adj[0][1]=-((\*(arr+1\*c+0)\*(\*(arr+2\*c+2)))-(\*(arr+1\*c+2)\*(\*(arr+2\*c+0)))); //determinant after deleting 1st row and 2nd column  adj[0][2]=(\*(arr+1\*c+0)\*(\*(arr+2\*c+1)))-(\*(arr+1\*c+1)\*(\*(arr+2\*c+0))); //determinant after deleting 1st row and 3rd column  adj[1][0]=-((\*(arr+0\*c+1)\*(\*(arr+2\*c+2)))-(\*(arr+2\*c+1)\*(\*(arr+0\*c+2)))); //determinant after deleting 2nd row and 1st column  adj[1][1]=(\*(arr+0\*c+0)\*(\*(arr+2\*c+2)))-(\*(arr+2\*c+0)\*(\*(arr+0\*c+2))); //determinant after deleting 2nd row and 2nd column  adj[1][2]=-((\*(arr+0\*c+0)\*(\*(arr+2\*c+1)))-(\*(arr+0\*c+1)\*(\*(arr+2\*c+0)))); //determinant after deleting 2nd row and 3rd column  adj[2][0]=(\*(arr+0\*c+1)\*(\*(arr+1\*c+2)))-(\*(arr+1\*c+1)\*(\*(arr+0\*c+2))); //determinant after deleting 3rd row and 1st column  adj[2][1]=-((\*(arr+0\*c+0)\*(\*(arr+1\*c+2)))-(\*(arr+1\*c+0)\*(\*(arr+0\*c+2)))); //determinant after deleting 3rd row and 2nd column  adj[2][2]=(\*(arr+1\*c+1)\*(\*(arr+0\*c+0)))-(\*(arr+0\*c+1)\*(\*(arr+1\*c+0))); //determinant after deleting 3rd row and 3rd column    **for**(**int** i=0; i<c; i++){  **for**(**int** j=0; j<c; j++){  cout<<adj[j][i]<<" "; //displaying adjoint matrix after taking its transpose  }  cout<<endl;  }  }  }  **double** inverse(**int** \*arr, **int** c) //function to find inverses  {  **double** l,m,n;  **double** det,adj[c][c],inverse1[c][c];    **if**(c==2)  {  //taking here determinant again because if user wants to know inverse first so he can  det=\*(arr+0\*c+0)\*(\*(arr+1\*c+1))-(\*(arr+0\*c+1))\*(\*(arr+1\*c+0));    //taking adjoint  adj[0][0]= \*(arr+1\*c+1); //to interchange values of diagonal  adj[1][1]=\*(arr+0\*c+0);  adj[0][1]=-(\*(arr+0\*c+1)); //multiplying non-diagonal entities with negative sign  adj[1][0]=-(\*(arr+1\*c+0));    **for**(**int** i=0; i<c; i++){  **for**(**int** j=0; j<c; j++){  inverse1[i][j]=(adj[i][j])/det;  }  }    cout<<"the inverse of the matrix is:"<<endl;  **for**(**int** i=0; i<c; i++){  **for**(**int** j=0; j<c; j++){  cout<<inverse1[i][j]<<" "; //for displaying values that are calculated by formula  }  cout<<endl;  }  }    **if**(c==3){    //first calculating determinant  l=(\*(arr+1\*c+1)\*(\*(arr+2\*c+2)))-(\*(arr+2\*c+1)\*(\*(arr+1\*c+2)));  m=(\*(arr+1\*c+0)\*(\*(arr+2\*c+2)))-(\*(arr+2\*c+0)\*(\*(arr+1\*c+2)));  n=(\*(arr+1\*c+0)\*(\*(arr+2\*c+1)))-(\*(arr+2\*c+0)\*(\*(arr+1\*c+1)));  det=(l\*(\*(arr+0\*c+0)))-(m\*(\*(arr+0\*c+1)))+(n\*(\*(arr+0\*c+2)));    //calculating adjoint  adj[0][0]=(\*(arr+1\*c+1)\*(\*(arr+2\*c+2)))-(\*(arr+2\*c+1)\*(\*(arr+1\*c+2))); //determinant after deleting 1st row and 1st column  adj[0][1]=-((\*(arr+1\*c+0)\*(\*(arr+2\*c+2)))-(\*(arr+1\*c+2)\*(\*(arr+2\*c+0)))); //determinant after deleting 1st row and 2nd column  adj[0][2]=(\*(arr+1\*c+0)\*(\*(arr+2\*c+1)))-(\*(arr+1\*c+1)\*(\*(arr+2\*c+0))); //determinant after deleting 1st row and 3rd column  adj[1][0]=-((\*(arr+0\*c+1)\*(\*(arr+2\*c+2)))-(\*(arr+2\*c+1)\*(\*(arr+0\*c+2)))); //determinant after deleting 2nd row and 1st column  adj[1][1]=(\*(arr+0\*c+0)\*(\*(arr+2\*c+2)))-(\*(arr+2\*c+0)\*(\*(arr+0\*c+2))); //determinant after deleting 2nd row and 2nd column  adj[1][2]=-((\*(arr+0\*c+0)\*(\*(arr+2\*c+1)))-(\*(arr+0\*c+1)\*(\*(arr+2\*c+0)))); //determinant after deleting 2nd row and 3rd column  adj[2][0]=(\*(arr+0\*c+1)\*(\*(arr+1\*c+2)))-(\*(arr+1\*c+1)\*(\*(arr+0\*c+2))); //determinant after deleting 3rd row and 1st column  adj[2][1]=-((\*(arr+0\*c+0)\*(\*(arr+1\*c+2)))-(\*(arr+1\*c+0)\*(\*(arr+0\*c+2)))); //determinant after deleting 3rd row and 2nd column  adj[2][2]=(\*(arr+1\*c+1)\*(\*(arr+0\*c+0)))-(\*(arr+0\*c+1)\*(\*(arr+1\*c+0))); //determinant after deleting 3rd row and 3rd column    **for**(**int** i=0; i<c; i++){  **for**(**int** j=0; j<c; j++){  inverse1[i][j]=(adj[j][i])/det; //taking transpose of cofactor matrix  }  }    cout<<"the inverse of the matrix is:"<<endl;  **for**(**int** i=0; i<c; i++){  **for**(**int** j=0; j<c; j++){  cout<<inverse1[i][j]<<" "; //for displaying values that are calculated by formula  }  cout<<endl;  }  }  **return** 0;  }  **void** solution(**int** \*arr,**int** w)  {  **double** bmatrix[w][1],D,Dx,Dy,Dz,a,b,c,d,e,f,x,y,z,g,h,i,j,k,l;  cout<<"enter matrix b"<<endl;  **for**(**int** i=0; i<w; i++){  **for**(**int** j=0; j<1; j++){  cin>>bmatrix[i][j]; //input of a new matrix needed for cramers rule  }  }  **if**(w==2){  a=\*(arr+0\*w+0);  b=\*(arr+0\*w+1);  c=\*(arr+1\*w+0);  d=\*(arr+1\*w+1);  e=bmatrix[0][0];  f=bmatrix[1][0];  D=(a\*d)-(b\*c); //determinant of original matrix  Dx=(e\*d)-(b\*f); //determinant after exchanging first column  Dy=(a\*f)-(e\*c); //determinant after exchanging second column  x=Dx/D; //formula of cramers rule  y=Dy/D;  cout<<"x="<<x<<endl<<"y="<<y<<endl;  }  **if**(w==3){  //matrixA values  a=\*(arr+0\*w+0);  b=\*(arr+0\*w+1);  c=\*(arr+0\*w+2);  d=\*(arr+1\*w+0);  e=\*(arr+1\*w+1);  f=\*(arr+1\*w+2);  g=\*(arr+2\*w+0);  h=\*(arr+2\*w+1);  i=\*(arr+2\*w+2);  //matrixb values  j=bmatrix[0][0];  k=bmatrix[0][0];  l=bmatrix[0][0];  //determinant of the 4 matrices  D=a\*((e\*i)-(h\*f))-b\*((d\*i)-(g\*f))+c\*((d\*h)-(g\*e));  Dx=j\*((e\*i)-(f\*h))-b\*((k\*i)-(f\*l))+c\*((k\*h)-(e\*l));  Dy=a\*((k\*i)-(f\*l))-j\*((d\*i)-(f\*g))+c\*((d\*l)-(k\*g));  Dz=a\*((e\*l)-(k\*h))-b\*((d\*l)-(k\*g))+j\*((d\*h)-(g\*e));  //formula of cramers rule  x=Dx/D;  y=Dy/D;  z=Dz/D;  cout<<"x="<<x<<endl<<"y="<<y<<endl<<"z="<<z<<endl;  }    }  **int** main()  {  **int** m,n;  **double** determ;  **unsigned** **int** num; //used unsigned int because it only takes positive numbers    cout<<"enter order of matrix"<<endl; //input of matrix no. of rows and columns  cin>>m>>n;    **if**(m==n){ //condition to make sure if it is a square matrix  **int** matrix[m][n];  cout<<"enter values of matrix"<<endl;  **for**(**int** i=0; i<m; i++){ //loop for row input of matrix  **for**(**int** j=0; j<n;j++){ //loop for column input of matrix  cin>>matrix[i][j];  }  }    cout<<"enter digit from 1-7"<<endl<<"✈︎to display press '1'"<<endl<<"✈︎to check if symmetric press '2'"<<endl<<"✈︎to check if identity matrix press '3'"<<endl<<"✈︎to find determinant press '4'"<<endl<<"✈︎to determine adjoint press '5'"<<endl<<"✈︎to determine inverse press '6'"<<endl<<"✈︎to determine the solution press '7'"<<endl<<"✈︎to exit press '8'"<<endl;    **for**(**int** i=0; i!='056'; i++) //using ascii code of 8. the loop will not terminate until 8 is entered  { //to take input from user till the user does not want to exit the program  cout<<"enter action number to be performed:";  cin>>num;  cout<<endl;    **if**(num>0 && num<8) //condition to check the correct option is entered  {    **switch**(num){ //switch statement to check which funtion to use  **case** 1:  //to display output  display((**int**\*)matrix, n) ; //function calling by using pointers  **break**;  **case** 2:  symmetric((**int**\*)matrix,n); //to check if it is a symmetric matrix  **break**;  **case** 3:  identity((**int**\*)matrix,n); //function call to check if it is an identity matrix  **break**;  **case** 4:  determ=determinant((**int**\*)matrix,n);  cout<<"the determinant is="<<determ<<endl; //function to calculate determinant of the matrix  **break**;  **case** 5:  adjoint((**int**\*)matrix,n); //function call to calculate the adjoint of the matrix  **break**;  **case** 6:  inverse((**int**\*)matrix,n); //function call to calculate invere of the matrix  **break**;  **case** 7:  solution((**int**\*)matrix,n); //function call solve the matrix  **break**;  }  }    **else** **if**(num==8){  **break**; //if user enters 8 it exits the program  }    **else**  cout<<"incorrect number entered"<<endl;  }  }    **else**  cout<<"only enter square matrix"<<endl;    **return** 0;  } |
| --- |

output of the code for 2x2 matrix:

| ➡️it is displaying the matrix  ➡️checking if symmetric  ➡️checking if identity  ➡️calculating determinant  ➡️determining adjoint  ➡️determining inverse  ➡️calculating cramers rule  ➡️exiting the program on clicking 8    to check if symmetric    to check if identity: |
| --- |

| for output of 3x3 matrix:    ➡️it is displaying the matrix  ➡️checking if symmetric  ➡️checking if identity  ➡️calculating determinant  ➡️determining adjoint  ➡️determining inverse  ➡️calculating cramers rule  ➡️exiting the program on clicking 8 |
| --- |

1. CLO 5: **Implement** structured approach in C++ programming by employing simple functions.

   LLO 1: **Develop** an understanding about debugging, data types, input/output statements, arithmetic, relational and logical operators using Dev C++.

   LLO 2: **Design** algorithms to solve problems using control structures and repetition statements with nesting.

   LLO 3: **Develop** a well-structured program using simple functions, char arrays, one-dimensional and two-dimensional static arrays. [↑](#footnote-ref-0)