**DATE:-**

**ASSIGNMENT NUMBER :-**

**PROBLEM STATEMENT:-**

Program in C to implement Gauss elimination method.

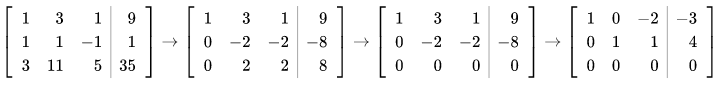
**THEORY:-**

In [linear algebra](https://en.wikipedia.org/wiki/Linear_algebra), Gaussian elimination (also known as row reduction) is an [algorithm](https://en.wikipedia.org/wiki/Algorithm) for solving [systems of linear equations](https://en.wikipedia.org/wiki/System_of_linear_equations). It is usually understood as a sequence of operations performed on the corresponding [matrix](https://en.wikipedia.org/wiki/Matrix_(mathematics)) of coefficients. This method can also be used to find the [rank](https://en.wikipedia.org/wiki/Rank_(linear_algebra)) of a matrix, to calculate the [determinant](https://en.wikipedia.org/wiki/Determinant) of a matrix, and to calculate the inverse of an [invertible square matrix](https://en.wikipedia.org/wiki/Invertible_matrix). The method is named after [Carl Friedrich Gauss](https://en.wikipedia.org/wiki/Carl_Friedrich_Gauss) (1777–1855), although it was known to Chinese mathematicians as early as 179 CE (see [History section](https://en.wikipedia.org/wiki/Gaussian_elimination#History)).

To perform row reduction on a matrix, one uses a sequence of [elementary row operations](https://en.wikipedia.org/wiki/Elementary_row_operations) to modify the matrix until the lower left-hand corner of the matrix is filled with zeros, as much as possible. There are three types of elementary row operations:

1. Swapping two rows.
2. Multiplying a row by a non-zero number.
3. Adding a multiple of one row to another row.

Using these operations, a matrix can always be transformed into an [upper triangular matrix](https://en.wikipedia.org/wiki/Triangular_matrix), and in fact one that is in [row echelon form](https://en.wikipedia.org/wiki/Row_echelon_form). Once all of the leading coefficients (the left-most non-zero entry in each row) are 1, and every column containing a leading coefficient has zeros elsewhere, the matrix is said to be in [reduced row echelon form](https://en.wikipedia.org/wiki/Reduced_row_echelon_form). This final form is unique; in other words, it is independent of the sequence of row operations used. For example, in the following sequence of row operations (where multiple elementary operations might be done at each step), the third and fourth matrices are the ones in row echelon form, and the final matrix is the unique reduced row echelon form.

{\displaystyle \left[{\begin{array}{rrr|r}1&3&1&9\\1&1&-1&1\\3&11&5&35\end{array}}\right]\to \left[{\begin{array}{rrr|r}1&3&1&9\\0&-2&-2&-8\\0&2&2&8\end{array}}\right]\to \left[{\begin{array}{rrr|r}1&3&1&9\\0&-2&-2&-8\\0&0&0&0\end{array}}\right]\to \left[{\begin{array}{rrr|r}1&0&-2&-3\\0&1&1&4\\0&0&0&0\end{array}}\right]}

**ALGORITHM:-**

**Input specification:** An array say **a[]**, which will contain the stack,

a variable, say top which will contain the top element of the stack and another variable say s, which will contain the size of the stack.

**Output specification:** Successful occurrence of the operations of the stack.

**Steps:**

**Algorithm for method ins():**

1. If(top==s-1) Then
   1. Print "Stack overflow"
2. Else
   1. Print "Enter the element to insert: "
   2. top=top+1
   3. Input a[top]

**Algorithm for method del():**

1. If(top<0) Then
   * 1. Print "Stack underflow"
2. Else
   1. Print "The deleted element is” a[top]
   2. top=top-1

Algorithm for method disp()

1. If(top<0)
   1. Print "Stack underflow"
2. Repeat Step 2.a to 2.b For i=top to i>=0
   1. Print a[i]
   2. i=i-1

**Algorithm for method main()**

1. Print "Enter the size of the stack: "
   1. Input s
2. While(TRUE)
   1. Print “1.Insertion 2.Deletion 3.Traverse 4.Exit”
   2. Print “Enter your choice: ");
   3. Input ch
   4. If(ch=1)Then
      1. ins()
      2. break
   5. Else If(ch=2)Then
      1. del()
      2. break
   6. Else If(ch=3)Then
      1. disp()
      2. break
   7. Else If(ch=4)Then
      1. Return
   8. Else
      1. Print "Wrong choice"
      2. Break

**SOURCE CODE:-**

#include<stdio.h>

int main()

{

    int i,j,k,n;

    float A[20][20],c,x[10],sum=0.0;

    printf("\nEnter the order of matrix: ");

    scanf("%d",&n);

    printf("\nEnter the elements of augmented matrix row-wise:\n\n");

    for(i=1; i<=n; i++)

    {

        for(j=1; j<=(n+1); j++)

        {

            printf("A[%d][%d] : ", i,j);

            scanf("%f",&A[i][j]);

        }

    }

    for(j=1; j<=n; j++) /\* loop for the generation of upper triangular matrix\*/

    {

        for(i=1; i<=n; i++)

        {

            if(i>j)

            {

                c=A[i][j]/A[j][j];

                for(k=1; k<=n+1; k++)

                {

                    A[i][k]=A[i][k]-c\*A[j][k];

                }

            }

        }

    }

    x[n]=A[n][n+1]/A[n][n];

    /\* this loop is for backward substitution\*/

    for(i=n-1; i>=1; i--)

    {

        sum=0;

        for(j=i+1; j<=n; j++)

        {

            sum=sum+A[i][j]\*x[j];

        }

        x[i]=(A[i][n+1]-sum)/A[i][i];

    }

    printf("\nThe solution is: \n");

    for(i=1; i<=n; i++)

    {

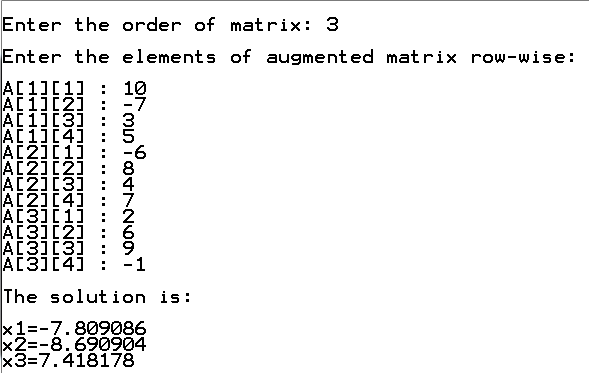
        printf("\nx%d=%f\t",i,x[i]); /\* x1, x2, x3 are the required solutions\*/

    }

    return(0);

}

**INPUT & OUTPUT:-**



**DISCUSSION:-**

1. You could compute the inverse and multiply with that. You can compute the inverse in (at least) two ways: with Gaussian elimination, or Kramer’s rule. The latter is very expensive and probably unstable. But if you already already have the LU factorization from Gaussian elimination, you might as well use that, rather than first compute the inverse, not?
2. More interesting is that you can use an iterative method for solving the linear system. In that case GE has the pro of being guaranteed to work (up to roundoff), while iterative methods can fail, or use an unpredictable amount of time. GE has the disadvantage in the practical case of sparse matrices that it needs way more memory, and potentially more time.

-----------------------------------