Basic Gauss Elimination Method

Steps:

1. Declare matrix & required variables
2. Enter Augmented Matrix
3. Forward Elimination
    a. Upper Triangular Matrix
4. Backward Substitution
5. Print Solution/Roots

Basic Equations:

$$3x + 2y + 1z = 10$$
$$2x + 3y + 2z = 14$$
$$1x + 2y + 3z = 14$$

x= 1, y=2, z=3

$$4x + 2y + 3z = 4$$
$$2x + 2y + z = 6$$
$$x + y + z = 0$$

x= 6, y=1, z=-6

$$3x + 2y - 4z + 3u = 2$$
$$2x + 3y - 3z - u = 1$$
$$x + 2y + 3z - u = 10$$
$$2x - y + 2z + 3u = 7$$

x= 1, y=2, z=2, u=1

Program Code in C

```
#include <stdio.h>
#include <conio.h>
#include <math.h>
int main(){
        int i, j, k;
        float pivot, factor,sum;
        int n=3; float x[3], a[3][4] ={{3,2,1,10},{2,3,2,14},{1,2,3,14}};
        // int n=3; float x[3], a[3][4] ={{4,2,3,4},{2,2,1,6},{1,1,1,0}};
        // int n=4; float x[4], a[4][5] ={{3,2,-4,3,2},{2,3,-3,-1,1},{1,2,3,-1,10},{2,-1,2,3,7}};
        clrscr();

        /* int n=10; float x[10], a[10][10];
        clrscr();
        printf("Enter the size of square matrix: ");
        scanf("%d", &n);
        fflush(stdin);
        printf("Enter the element of a[i][j]:\n");
        for(i=0; i<n; i++){
                for(j=0; j<n+1; j++){
                        scanf("%f", &a[i][j]);
                }
        }
        fflush(stdin);*/
```

```
/* Augmented Matrix */
printf("\nAugmented Matrix:\n");
for(i=0;i<n;i++){
        for(j=0;j<n+1;j++){
           if(j==(n-1)){
              printf("%f : ",a[i][j]);
           }else{
              printf("%f\t",a[i][j]);
           }
        }
        printf("\n");
}
```

```
/* Forward elimination */
for(j=0; j<n; j++){
   for(i=0;i<n;i++){
      if(i>j){
         factor = a[i][j]/a[j][j];
         for(k=0;k<n+1;k++){
            a[i][k]=a[i][k]-factor*a[j][k];
            printf("a[%d][%d] = %f\n",i,k,a[i][k]);
         }
      }
   }
}
/*for(k=0; k<n-1; k++){
        pivot=a[k][k];
        for(i=k+1; i<n; i++){
                factor=a[i][k]/pivot;
                for(j=k; j<n+1; j++){
                        a[i][j]= a[i][j] - factor*a[k][j];
                        printf("a[%d][%d] = %f\n",i,j,a[i][j]);
                }
        }
}*/
```

```
        /* Upper Triangular Matrix */
        printf("\nUpper Triangular Matrix:\n");
        for(i=0;i<n;i++){
                for(j=0;j<n+1;j++){
                   if(j==(n-1)){
                      printf("%f : ",a[i][j]);
                   }else{
                      printf("%f\t",a[i][j]);
                   }
                }
                printf("\n");
        }

/*Backward Substitution*/
x[n-1]= a[n-1][n]/a[n-1][n-1];
         /*for(i=n-2; i>=0; i--){
               sum=0;
               for(j=i+1; j<n; j++){
                  sum=sum+a[i][j]*x[j];
               }
               x[i]=(a[i][n]-sum)/a[i][i];
            }*/
        printf("\nz = %f/%f = %f",a[n-1][n],a[n-1][n-1],x[n-1]);
        for(k=n-2; k>=0; k--){
                sum=0.0;
                for(j=k+1; j<n; j++){
                        sum = sum + a[k][j]*x[j];
                printf("\nsum + (a[%d][%d]: %f) * (x[%d]: %f) = %f",k,j,a[k][j],j,x[j],sum);
                }
                x[k]=(a[k][n] - sum)/a[k][k];
printf("\nx[%d] = (a[%d][%d]:  %f - sum: %f)/ (a[%d][%d]: %f) = f",k,k,n,a[k][n],sum,k,k,x[k]);
        }

        /* Print Roots/Solution */
        printf("\n\nRoots of given equation are:\n");
        for(i=0; i<n; i++){
                printf("x[%d]= %f\n", i, x[i]);
        }
        getch();
   return 0;
}
```

Output

```
Augmented Matrix:
3.000000      2.000000      1.000000 : 10.000000
2.000000      3.000000      2.000000 : 14.000000
1.000000      2.000000      3.000000 : 14.000000
a[1][0] = 0.000000
a[1][1] = 1.666667
a[1][2] = 1.333333
a[1][3] = 7.333333
a[2][0] = 0.000000
a[2][1] = 1.333333
a[2][2] = 2.666667
a[2][3] = 10.666666
a[2][0] = 0.000000
a[2][1] = 0.000000
a[2][2] = 1.600000
a[2][3] = 4.800000

Upper Triangular Matrix:
3.000000      2.000000      1.000000 : 10.000000
0.000000      1.666667      1.333333 : 7.333333
0.000000      0.000000      1.600000 : 4.800000

z = 4.800000/1.600000 = 3.000000
sum + (a[1][2]: 1.333333) * (x[2]: 3.000000) = 3.999999
x[1] = (a[1][3]:  7.333333 - sum: 3.999999)/ (a[1][1]: 2.000000) = 3.999999
sum + (a[0][1]: 2.000000) * (x[1]: 2.000000) = 4.000001
sum + (a[0][2]: 1.000000) * (x[2]: 3.000000) = 7.000000
x[0] = (a[0][3]:  10.000000 - sum: 7.000000)/ (a[0][0]: 1.000000) = 7.000000

Roots of given equation are:
x[0]= 1.000000
x[1]= 2.000000
x[2]= 3.000000
```