# Classifications of Determinantal Varieties


*By*

SANA ZAHOOR

CIIT/FA17-RMT-054/LHR

M.S Thesis

In

Mathematics

COMSATS University Islamabad,

Lahore Campus

Spring, 2019


COMSATS University Islamabad, Lahore Campus

# Classifications of Determinantal Varieties

A Thesis Presented to

COMSATS University Islamabad,

Lahore Campus

In partial fulfillment

of the requirement for the degree of

M.S (Mathematics)

By

Sana Zahoor

CIIT/FA17-RMT-054/LHR

Spring, 2019

# Classifications of Determinantal Varieties

A Post Graduate Thesis submitted to the department of Mathematics as partial fulfillment of the requirement for the award of Degree of M.S in Mathematics.


| Name | Registeration Number |
|---|---|
| Sana Zahoor | CIIT/FA17-RMT-054/LHR |


**Supervisor**


Dr. Imran Ahmed

Assistant Professor

Department of Mathematics,

Lahore Campus.

COMSATS University Islamabad, Lahore Campus.


June, 2019.

## Final Approval

This thesis titled

# Classifications of Determinantal Varieties

*By*

SANA ZAHOOR

CIIT/FA17-RMT-054/LHR

Has been approved

For the COMSATS University Islamabad, Lahore Campus

**External Examiner:** _____

**Supervisor:** _____

Dr. Imran Ahmed

Department of Mathematics /CUI, Lahore Campus

**HOD:** _____

Dr. Sarfraz Ahmad

Department of Mathematics /CUI, Lahore Campus

# Declaration

I Sana Zahoor (CIIT/FA17-RMT-054/LHR) hereby declare that I have produced the work presented in this thesis, during the scheduled period of study. I also declare that I have not taken any material from any source except referred to wherever due that amount of plagiarism is within acceptable range. If a violation of HEC rules on research has occurred in this thesis. I shall be liable to punishable action under the plagiarism rules of the HEC.

Date:_____ Signature: _____

Sana Zahoor

CIIT / FA17-RMT-054/LHR

# Certificate

It is certified that Sana Zahoor with registration number CIIT/FA17-RMT-054/LHR has carried out all the work related to this thesis under my supervision at the Department of Mathematics, COMSATS University Islamabad, Lahore campus and the work fulfills the requirement for award of M.S degree.

Date:_____

Supervisor:

_____

Dr. Imran Ahmed
Assistant Professor
Department of Mathematics
CUI, Lahore Campus

Head of Department:

_____

Dr. Sarfraz Ahmad
Associate Professor
CUI, Lahore Campus

**DEDICATED**

**To**

**My loving parents,**

**siblings, friends and teachers**

# ACKNOWLEDGEMENTS

All praises to ALLAH Almighty who created the universe and knows whatever is in it hidden or evident, who bestowed upon me the intellectual ability and wisdom to search for it's secrets and whose blessings flourished my thoughts and thrived my ambitions and enabled me to contribute sincerely to the sacred wealth of knowledge. Special praise and regards to his last messenger Muhammad (PBUH) who is a tower of guidance and knowledge for humanity.

First of all, I pay my gratitude to Allah Almighty, the most Gracious and the most Merciful, Who helped me and gave me the courage and strength to complete my thesis. My very humble praise is to the beloved Holy Prophet Muhammad S.A.W whose love is light to me. I feel much pleasure in expressing my heartiest gratitude to my ever affectionate supervisor Dr. Imran Ahmed for his dynamic and friendly supervision. Whose indispensable guidance, deep consideration, constructive comments and active cooperation enabled me to complete my work successfully. His excellent knowledge, motivation and help combined to make my research experience worthwhile. All the words I know are insufficient to pay my tribute to him.

I would like to thank to Head of Department Dr. Sarfraz Ahmad, M.S Coordinator Dr. Abdul Jawad and all my teachers for their suggestions, guidance and constant support.

Of course, at this stage I cannot forget my whole family especially my respectable parents, my siblings especially my younger brother M.Kashif for their continuing support and prayers. Their prayers have lightened up my spirit to finish this thesis.

COMSATS University Islamabad, Lahore Campus.

**Sana Zahoor**

**CIIT/FA17-RMT-054/LHR**

# Abstract

Let $M_{ns}$ be a space of matrices of order $n \times s$ with complex enteries. The determinantal singularities are defined by vanishing of all the minors of a certain size of an $n \times s$ matrix $\alpha : \mathbb{C}^k \to M_{ns}$. The class of determinantal singularities is a more general class than the class of complete intersection singularities. The $\mathcal{G}$-finite deteminacy gives a special class of determinantal varieties known as essentially isolated determinantal singularities (EIDS), firstly introduced by Ebeling and Gusein-Zade in 2009. We give a computational method to find linear forms of EIDS by programming in C.

# Table of Contents

# Chapter 1

# Introduction

Let $M_{ns}$ be a space of matrices of order $n \times s$ with complex enteries. Suppose that $M_{ns}^r$ be a subset of $M_{ns}$ consists of matrices of rank less than $r$ with $1 \leq r \leq \min\{n, s\}$. It can be prove that $M_{ns}^r$ is an irreducible singular algebraic variety of codimension $(n-r+1)(s-r+1)$, see [3]. The singular locus of $M_{ns}^r$ coincides with $M_{ns}^{r-1}$. The singular locus of $M_{ns}^{r-1}$ is $M_{ns}^{r-2}$ etc, see [2]. The representation of the variety $M_{ns}^r$ as the union of $M_{ns}^j \backslash M_{ns}^{j-1}$, $j = 1, ..., r$, is a Whitney stratification of $M_{ns}^r$.

Let $\alpha : U \subset \mathbb{C}^k \to M_{ns}$, $\alpha(z) = (\gamma_{ij}(z))$ is an $n \times s$ matrix for each element $z \in U \subset \mathbb{C}^k$ such that $\gamma_{ij}$ are differentiable complex functions on an open domain $U$ of $\mathbb{C}^k$. The determinantal variety $Y$ is the preimage of the variety $M_{ns}^r$ given by $\mathrm{Y} = \alpha^{-1}(M_{ns}^r) = \gamma^{-1}(0)$ such that codimension of $Y$ is $(n-r+1)(s-r+1)$ in $\mathbb{C}^k$. We denote by $\gamma$ the map defined by $r \times r$ minors of $\alpha$. It can be proved that if the determinantal variety $Y$ has isolated singularity at the origin, then $k \leq (n-r+2)(s-r+2)$, see [6].

The essentially isolated deteminantal singularity (EIDS) is defined by Ebeling and Gusein-Zade in [6], see (Definition 2.0.1). Pereira[8,9] proved that for all $1 \leq r \leq min\{n, s\}$ the determinantal variety $Y^r = \alpha^{-1}(M_{ns}^r)$ is an EIDS if and only if $\alpha : (\mathbb{C}^k, 0) \to M_{ns}$ is $\mathcal{G}$-finitely determined. In [1], authors proved that $\mathcal{G}$-finite determinacy holds in general.

In this thesis, we give a computational method to find linear forms of EIDS by programming in C.

# Chapter 2

# Preliminaries

Suppose that $M_{ns}$ be a space of matrices of order $n \times s$ with complex enteries in $\mathbb{C}^{ns}$. Let a subspace $M_{ns}^r = \{A \in M_{ns} | rank(A) < r\}$ of space $M_{ns}$ having rank less than $r$ with $1 \leq r \leq \min\{n, s\}$. The set $M_{ns}^r$ is a generic determinantal variety if following conditions are satisfied:

$(i)$ The subspace $M_{ns}^r$ has codimension $(n - r + 1)(s - r + 1)$ in $M_{ns}$.

$(ii)$ The singular set of $M_{ns}^r$ coexist with $M_{ns}^{r-1}$. The singular set of $M_{ns}^{r-1}$ is $M_{ns}^{r-2}$ etc.

$(iii)$ The variety of $M_{ns}^r$ can be written as

$$M_{ns}^r = \coprod_{j=1,\ldots,r} (M_{ns}^j \backslash M_{ns}^{j-1}).$$

This partition is known as Whitney stratification of $M_{ns}^r$.

We define now essentially isolated deteminantal singularity (EIDS) as follows.

**Definition 2.0.1.** *[6] " A germ $(Y, 0) \subset (\mathbb{C}^k, 0)$ of a determinatal variety of type $(n, s, r)$ has an essentially isolated determinantal singularity at the origin if $\alpha : \mathbb{C}^k \to M_{ns}$ is transverse to all strata $M_{ns}^r \backslash M_{ns}^{r-1}$ of the stratification in the punctured neighbourhood of the origin."*

Consider a ring $\mathcal{O}_k$ with all differentiable functions in $\mathbb{C}^k$ and a maximal idea $\mathcal{M}$ of $\mathcal{O}_k$. We denote $M_{ns}(\mathcal{O}_k)$ the set of all matrices of order $n \times s$ with entries in $\mathcal{O}_k$. Let $\mathcal{G} = \mathcal{R} \ltimes \mathcal{H}$ is the semi direct product of $\mathcal{R}$ and $\mathcal{H}$ , where $\mathcal{R}$ is a group of isomorphisms of smooth manifolds and $\mathcal{H} = \mathrm{GL}_r(\mathcal{O}_k) \times \mathrm{GL}_s(\mathcal{O}_k)$, where $\mathrm{GL}_r(\mathcal{O}_k)$ is an invertible group of $r \times r$ matrices with entries in $\mathcal{O}_k$ and $\mathrm{GL}_s(\mathcal{O}_k)$ is an invertible group of $s \times s$ matrices with entries in $\mathcal{O}_k$.

**Definition 2.0.2.** *[1] The two germs $\alpha_1$ and $\alpha_2 \in M_{ns}(\mathcal{O}_k)$ are said to be $\mathcal{G}$ - equivalent if there exist $\phi \in \mathcal{R}$, $R \in \mathrm{GL}_r(\mathcal{O}_k)$, $L \in \mathrm{GL}_s(\mathcal{O}_k)$ , such that*

$$\alpha_1 = \mathrm{L}^{-1}(\varphi^* \alpha_2)\mathrm{R}$$

Given a matrix $\alpha \in M_{ns}(\mathcal{O}_k)$, let $C_{pq}(\alpha)$ be the matrix having $p$th column equal to the $q$th column of a matrix $\alpha$ and zeros elsewhere. Let $R_{uv}(\alpha)$ the matrix having $u$th

row equal to the $v$th row of a matrix $\alpha$ and zeros elsewhere. $\mathcal{J}(\alpha)$ is the submodule of $M_{ns}(\mathcal{O}_k)$, which is produced by matrices of the form $\frac{\partial \alpha}{\partial x_\eta}$ with $\eta = 1, ..., k$.
The $\mathcal{G}-$tangent space to a germ $\alpha$ is given by

$$\mathcal{TG}\alpha = \mathcal{MJ}(\alpha) + \mathcal{O}_k\{R_{uv}(\alpha), C_{pq}(\alpha)\},$$

where $1 \leq u, v \leq n$, $1 \leq p, q \leq s$, see [4,8].
The $\mathcal{G}$-codimension of $\alpha$ is given by

$$d(\alpha, \mathcal{G}) = dim_{\mathbb{C}} \frac{M_{ns}(\mathcal{O}_k)}{\mathcal{TG}\alpha}$$

The extended tangent space and extended $\mathcal{G}$- codimension is given by

$$\mathcal{TG}_e\alpha = \mathcal{J}(\alpha) + \mathcal{O}_k\{R_{uv}(\alpha), C_{pq}(\alpha)\}$$

$$d_e(\alpha, \mathcal{G}) = dim_{\mathbb{C}} \frac{M_{ns}(\mathcal{O}_k)}{\mathcal{TG}_e\alpha}$$

Let $\underline{w} = (w_1, ..., w_k) \in \mathbb{N}^k$ be a set of weights . For any monomial $x^\beta = x_1^{\beta_1}, ..., x_k^{\beta_k}$, the $\underline{w}$ filtration of $x^\beta$ can be defined as

$$fil(x^\beta) = \sum_{i=1}^{k} w_i\beta_i$$

A filtration on the ring $\mathcal{O}_k$ can be defined as:

$$fil(g) = inf_\beta \left\{ fil(x^\beta)\} | \frac{\partial^{|\beta|} f}{\partial x^\beta}(0) \neq 0 \right\}$$

for all germs $f \in \mathcal{O}_k$, where$|\beta| = \beta_1 + \beta_1 + ... + \beta_k$.

**Definition 2.0.3.** *[1] We define $fil(\alpha) = \mathcal{D} = d_{ij}$, where $d_{ij} = fil(\gamma_{ij})$ for $1 \leq i \leq n$, $1 \leq j \leq s$. Then a matrix $\alpha \in M_{ns}(\mathcal{O}_k)$ is said to be a weighted homogeneous if $fil(\gamma_{ij}) = d_{ij}$ with respect to $\underline{w} = (w_1, ..., w_k)$ such that following conditions are satisfied:*

$$d_{ij} - d_{iv} = d_{uj} - d_{uv} \text{ where } 1 \leq i, u \leq n, 1 \leq j, v \leq s$$

**Definition 2.0.4.** *[7] "The matrix*

$$\alpha : (\mathbb{C}^k, 0) \to M_{ns}(\mathcal{O}_k)$$

*is* h-$\mathcal{G}$*-determined if for every* $\mathrm{G} \in M_{ns}(\mathcal{O}_k)$, $j^h\mathrm{G}(0) = j^h\alpha(0)$, *then* $\alpha$ *and* $G$ *are* $\mathcal{G}$*-equivalent. If such* $h$ *exists, we say that* $\alpha$ *is* $\mathcal{G}$*-finitely determined."*

These following Theorems have been proved in [9].

**Theorem 2.0.5.** *[9] "The followning conditions are equivalent.*
*(i)* $\alpha \in M_{ns}(\mathcal{O}_k)$ *is* $\mathcal{G}$*-finitely determined.*
*(ii)* $T\mathcal{G}\alpha \supset \mathcal{M}_k^p M_{ns}(\mathcal{O}_k)$ *for some positive integer* $p$.
*(iii)* $d_e(\alpha, \mathcal{G}) < \infty$."

**Theorem 2.0.6.** *[9] "The followning conditions are equivalent.*
*(i)* $\alpha : (\mathbb{C}^k, 0) \to M_{ns}$ *is* $\mathcal{G}$*-finitely determined.*
*(ii)* $Y^r = \alpha^{-1}(M_{ns}^r)$ *is an EIDS for all* $1 \leq r \leq min\{n, s\}$."

The following propositions in [1] imply that $\mathcal{G}$-determinacy holds in general.

**Proposition 2.0.7.** *[1] "We assign positive integer weights* $w_t$ *to* $z_t$, $t = 1, ..., k$. *Suppose that* $\gamma_{ij}(z_1, ..., z_k)$ *is weighted homogeneous of degree* $d_{ij}$, $1 \leq i \leq n$, $1 \leq j \leq s$, *with respect to weights* $(w_1, ..., w_k)$. *Consider* $\alpha : (\mathbb{C}^k, 0) \to M_{ns}$ *is finitely* $\mathcal{G}$*-determined and weighted homogeneous of type* $(\mathcal{D}; w)$. *Let* $G : (\mathbb{C}^k, 0) \to M_{ns}$ *be a polynomial mapping* $G = (g_{ij})$ *such that degree* $g_{ij} < d_{ij}$. *Then* $\alpha + G : (\mathbb{C}^k, 0) \to M_{ns}$ *is finitely* $\mathcal{G}$*-determined.Moreover,*

$$d_e(\alpha, \mathcal{G}) \geq d_e(\alpha + G, \mathcal{G})\text{"}$$

**Proposition 2.0.8.** *[1] "Let* $\mathrm{L}_{(b,\eta)}(z)$ *be the* $n \times s$ *matrix*

$$\mathcal{L}_{(b(i,j),\eta)}(z) = \sum_{\eta=1}^{k} b(i,j)_\eta z_\eta^d$$

*homogeneous polynomials of degree* $d \in \mathbb{N}^*$, *with* $1 \leq i \leq n$; $1 \leq j \leq s$; $1 \leq \eta \leq k$. *Then, for almost all values of the parameters* $b = (b(i,j))_\eta L_{(b,\eta)} : (\mathbb{C}^k, 0) \to M_{ns}$ *is* $\mathcal{G}$*-finitely determined"*

**Theorem 2.0.9.** *[1] "Finite $\mathcal{G}$-determinacy holds in general in $M_{ns}(\mathcal{O}_k)$."*

For a Cohen-Macaulay codimension 2 determinantal variety with isolated singularity, the following result was proved in [8,9].

**Theorem 2.0.10.** *[9] "Let $\alpha : (\mathbb{C}^k, 0) \to M_{(s+1)s}$ define a Cohen-Macaulay codimension 2 isolated singular variety $Y = \alpha^{-1}(M^s_{(s+1)s})$. Let $\gamma$ be the map defined by $r \times r$ minors of $\alpha$. Then the following conditions are equivalent.*
*(i) $Y \cap V(\mathcal{J}(\gamma)) = \{0\}$, where $\mathcal{J}(\gamma)$ is the ideal generated by the $2 \times 2$ minors of the Jacobian matrix of $\gamma$.*
*(ii) $\mathcal{J}(\gamma) + (\gamma_1, ..., \gamma_{s+1}) \supset \mathcal{M}^p_k$, for some positive integer $p$.*
*(iii) $\alpha$ is $\mathcal{G}$-finitely determined."*

# Chapter 3

# Linear Normal Forms of EIDS

## 3.1 Computational Method of Finding Linear EIDS

**Let $\alpha : \mathbb{C}^k \to M_{ns}$ be a matrix of order $n \times s$ with k variables. To get some normal forms for k=9,8,7,6,5 of linear EIDS given below, we use computer programme, see Table 3.2.**

Table 3.1: Normal forms of Linear EIDS
k is for no.of variables

| Ex no. | k | $M_{3\times3}$ |
|---|---|---|
| 1 | 9 | $\begin{pmatrix} u_1 & u_2 & u_3 \\ u_4 & u_5 & u_6 \\ u_7 & u_8 & u_9 \end{pmatrix}$ |
| 2 | 8 | $\begin{pmatrix} u_1 & u_2 & u_3 \\ u_4 & u_5 & u_6 \\ u_7 & u_1+u_6 & u_8 \end{pmatrix}$ |
| 3 | 7 | $\begin{pmatrix} u_1 & u_2 & u_3 \\ u_4 & u_5 & u_6 \\ u_7 & u_1+u_6 & -u_5 \end{pmatrix}$ |
| 4 | 6 | $\begin{pmatrix} u_1 & u_2 & u_3 \\ u_4 & u_5 & u_6 \\ u_3+u_4 & u_1+u_6 & -u_5 \end{pmatrix}$ |
| 5 | 5 | $\begin{pmatrix} u_1 & u_2 & u_3 \\ u_4 & u_5 & u_1 \\ u_3+u_4 & u_1 & -u_5 \end{pmatrix}$ |

We give method to find normal forms of linear EIDS by programming in C.

Table 3.2: C Language Computer Programming Code

**C Language Computer Programming Code**

```
1. #include<stdio.h>
2. #include<stdlib.h>
3. #include<string.h>
4. #include<stdbool.h>
5. #include<malloc.h>
6. typedef struct STRING{
7.     char field[100];
8. }String;
9. int x,y;
10. int eq_count=0;
11. int col_count=0;
12. String* cols;
13. void display_Matrix(String matrix[x][y]){
14.     printf("\n");
15.      for(int i=0;i<x;i++){
16.         for(int j=0;j<y;j++){
17.             printf("%10s\t",matrix[i][j].field);
18.         }
19.        printf("\n");
20.     }
21. }
22. int len(char *str){
23.     int i=0;
24.     while(str[i]!='\0'){
25.         i++;
26.     }
27. return i;
28. }
29. void derivative2(String matrix[x][y], char *str,
      String *variables, int v){
30.     for(int i=0;i<x;i++){//taking no. of rows
31.         for(int j=0;j<y;j++){//taking no. of columns
32.             for(int u=0;u<v;u++){//taking variable's position
```

Table 3.3: C Language Computer Programming Code

**C Language Computer Programming Code**

```
33.    char *t;
34.  if((t=strstr(variables[u].field, str))!=NULL){
35.  replace_string(matrix[i][j].field, str, "1");
36.                 }
37.    else{
38.    char *temp;
39.   while(((temp=strstr(matrix[i][j].field, variables[u].field))!= NULL)
       && (isdigit(*(temp+len(variables[u].field))) != 1)){
40.   replace_string(matrix[i][j].field,variables[u].field,"00");
41.                     }
42.                 }
43.
44.             }
45.    char *t;
46.   while((t=strstr(matrix[i][j].field, "u")) != NULL){
47.                 char c[5];
48.                 int p=1;
49.                 c[0]='u';
50.                 while(isdigit(*(t+1))){
51.                     c[p]=*(t+1);
52.                     c[p+1]='\0';
53.                     t++;
54.                     p++;
55.                 }
56.   replace_string(matrix[i][j].field, c,"00");
57.             }
58.         }
59.     }
60. }
61. void derivative3(String matrix[x][y],
        char *str, String *variables, int v){
62. for(int i=0;i<x;i++){//taking no. of rows
63.   for(int j=0;j<y;j++){//taking no. of columns
64.   for(int u=0;u<v;u++){//taking variable's position
65.    char *t;
66. if((t=strstr(variables[u].field, str))!=NULL){
67.   replace_string(matrix[i][j].field, str, "");
68.                 }
69.    else{
70.    char *temp;
71.   while(((temp=strstr(matrix[i][j].field, variables[u].field))!= NULL)
       && (isdigit(*(temp+len(variables[u].field))) != 1)){
72.   replace_string(matrix[i][j].field,variables[u].field,"00");
73.                     }
74.                 }
75.             }
```

Table 3.4: C Language Computer Programming Code

**C Language Computer Programming Code**

```
76.    char *t;
77.   while((t=strstr(matrix[i][j].field , "u")) != NULL){
78.                 char c[5];
79.                 int p=1;
80.                 c[0]='u';
81.                 while(isdigit(*(t+1))){
82.                     c[p]=*(t+1);
83.                     c[p+1]='\0';
84.                     t++;
85.                     p++;
86.                 }
87.                 replace_string(matrix[i][j].field , c,"00");
88.             }
89.         }
90.     }
91. }
//In Str replace str1 by str2
92. void replace_string(char *str,char *str1,char* str2){
93.     char *temp;
94.     if((temp=strstr(str,str1))!=NULL){
95.         int s=0;
96.         int s1=0;
97.         int s2=0;
98.         while(str[s]!='\0'){
99.             s++;
100.         }
101.         while(str1[s1]!='\0'){
102.             s1++;
103.         }
104.         while(str2[s2]!='\0'){
105.             s2++;
106.         }
107.         int ts=s+s2-s1;
108.             if(ts>s){
109.             while(&str[s]!=temp){
110.                 str[ts]=str[s];
111.                 s--;
112.                 ts--;
113.             }
114.             int m=0;
115.             while(m<s2){
116.                 *(temp+m)=str2[m];
117.                 m++;
118.             }
119.         }
```

Table 3.5: C Language Computer Programming Code

**C Language Computer Programming Code**

```
120.         else if(ts<s){
121.              int x=s2;
122.              char *ptr=(temp+s1);
123.              while(s1<=s){
124.                   *(temp+s2)=*(temp + s1);
125.                   s2++;
126.                   s1++;
127.              }
128.              int m=0;
129.              while(m<x){
130.     *(temp+m)=str2[m];
131.                 m++;
132.             }
132.       }
133.         else{
134.              int m=0;
135.              while(m<s2){
136.                   *(temp+m)=str2[m];
137.                 m++;
138.              }
139.         }
140.      }
141. }
142.  void copy_matrix(String matrix[x][y], String mat[x][y]){
143.    for(int i=0;i<x;i++){
144.         for(int j=0;j<y;j++){
145.              strcpy(mat[i][j].field , matrix[i][j].field);
146.         }
147.      }
148. }
149. void formEquation(String mat[x][y], String *equation,
         String *variables, int v){
150.    String eMat[x][y];
151.    int count=0;
152.    for(int i=0;i<x;i++){
153.         for(int j=0;j<y;j++){
154.        count++;
155.    strcpy(eMat[i][j].field , "e");
156.              char cc[4];
157.               itoa(count, cc, 10);
158.              strcat(eMat[i][j].field, cc);
159.         }
160.    }
```

Table 3.6: C Language Computer Programming Code

**C Language Computer Programming Code**

```
//display_Matrix(mat);
161. for(int i=0;i<x;i++){//replace the 00+00 with 00
162.         for(int j=0;j<y;j++){
163.   if(strcmp(mat[i][j].field,"00") == 0
          || strcmp(mat[i][j].field,"00+00") ==0
          ||strcmp(mat[i][j].field,"00-00") ==0
          || strcmp(mat[i][j].field,"00-00-00") ==0
          || strcmp(mat[i][j].field,"00-00+00") ==0
          || strcmp(mat[i][j].field,"00+00-00") ==0
          || strcmp(mat[i][j].field,"00+00+00") ==0){
164.                 strcpy(eMat[i][j].field,"00");
165.             }
166.             else{
167.             delete_zeros(mat[i][j].field);
168.             if(strcmp(mat[i][j].field , "00")==0){
169.                 strcpy(eMat[i][j].field, "");
170.             }
171.                 else{
172.                         multiplay(mat[i][j].field, eMat[i][j].field);
173.                 }
174.             }
175.         }
176.     }
//Now from matrix to equation
177.     char expression[200]="";
178.     for(int i=0;i<x;i++){
179.   for(int j=0;j<y; j++){
180.   if(strcmp(eMat[i][j].field,"00")!=0){
181.                 if(i!=0 && j!=0 && eMat[i][j].field[0]=='-' ){
182.                     }
183.                 else if((i!=0 || j!=0) && len(expression)>1){
184.                     strcat(expression,"+");
185.                 }
186.                 strcat(expression, eMat[i][j].field);
187.         }
188.     }
189. }
//multiplying with variables
190.     for(int i=0;i<v;i++){
191.         char str[200]="";
192.         strcat(str, variables[i].field);
193.         multiplay(expression, str);
194.         strcpy(equation[eq_count].field, str);
195.         eq_count=eq_count +1;
196.     }
197. }
```

Table 3.7: C Language Computer Programming Code

**C Language Computer Programming Code**

```
198. void formDerivateEq(String mat[x][y], String *equation,
       String *variables, int v){
199.    String eMat[x][y];
200.    int count=0;
201.    for(int i=0;i<x;i++){
202.    for(int j=0;j<y;j++){
203.  count++;
204.   strcpy(eMat[i][j].field , "e");
205.    char cc[4];
206.        itoa(count, cc, 10);
207.            strcat(eMat[i][j].field, cc);
208.        }
209.    }
//display_Matrix(mat);
210.    for(int i=0;i<x;i++){//replace the 00+00 with 00
211.     for(int j=0;j<y;j++){
212.  if(strcmp(mat[i][j].field,"00") == 0
          || strcmp(mat[i][j].field,"00+00") ==0
          ||strcmp(mat[i][j].field,"00-00") ==0
          || strcmp(mat[i][j].field,"00-00-00") ==0
          || strcmp(mat[i][j].field,"00-00+00") ==0
          || strcmp(mat[i][j].field,"00+00-00") ==0
          || strcmp(mat[i][j].field,"00+00+00") ==0){
213.                strcpy(eMat[i][j].field,"00");
214.            }
215.            else{
216.            delete_zeros(mat[i][j].field);
217.            if(strcmp(mat[i][j].field , "00")==0){
218.                strcpy(eMat[i][j].field, "");
219.            }
220.                else{
221.  multiplay(mat[i][j].field, eMat[i][j].field);
222.                }
223.            }
224.        }
225.    }
//Now from matrix to equation
226.    char expression[200]="";
227.     for(int i=0;i<x;i++){
228.        for(int j=0;j<y; j++){
229.     if(strcmp(eMat[i][j].field,"00")!=0){
230.    if(eMat[i][j].field[0] =='-'){
231.     strcat(expression,"-");
232.                }
```

Table 3.8: C Language Computer Programming Code

**C Language Computer Programming Code**

```
233.       else if(len(expression) > 1){
234.      strcat(expression,"+");
235.                 }
236.  strcat(expression, eMat[i][j].field);
237.         }
238.     }
239.}
//multiplying with variables
240.    if(strcmp(expression, "")==0 || expression == NULL){
241. }
242.  else{
243.     for(int i=0;i<v;i++){
244.         char str[200]="";
245.         strcat(str, variables[i].field);
246.         multiplay(expression, str);
247.    for(int j=i; j<v; j++){
248.       char dup[200];
249.       strcpy(dup,str);
250.      char cpy[200];
251.  strcpy(cpy,variables[j].field);
252.       multiplay2(dup,  cpy);
253.    strcpy(equation[eq_count].field, cpy);
254.  eq_count=eq_count + 1;
255.                                    }
256.                                }
257.                            }
258.                 }
259. void multiplay(char *s, char *str){
260.     int x=0;
261.     char expression[100];
262.    strcpy(expression,"");
263.     char eq[100]={""};
264.     strcpy(eq,s);
265.     int i=0;
266.     if(eq[0] != '-'){
267.         char t[100]={""};
268.    strcpy(t,str);
269.     strcat(t,eq);
270. strcpy(eq,t);
271.         i=1;
272.     }
273.         char *copy=strdup(eq);
274.         char *delim="+-";
```

Table 3.9: C Language Computer Programming Code

**C Language Computer Programming Code**

```
275.         char *tok=strtok(eq, delim);
276.         while(tok!=NULL){
277.              char temp[100]={""};
278.             if(x==0){
279.                 strcpy(temp, "-");
280.             }
281.             else{
282.        strcpy(temp, "+");
283.             }
284.              if(i != 1){
285.                  strcat(temp , str);
286.                  strcat(temp, tok);
287.                 strcat(expression, temp);
288.             }
289.             else{
290.                  strcat(expression, tok);
291.              }
292.       int from = tok-eq+strlen(tok);
293.       tok=strtok(NULL,delim);
294.     int to = tok != NULL ? tok-eq : strlen(copy);
295.             char c[4];
296.             sprintf(c, "%.*s", to-from, copy+from);
297.          if(c[0] == '+'){
298.            x=1;
299.                  }
300.                  else if(c[0] == '-'){
301.                      x=0;
302.                  }
303.              i--;
304.          }
305.          strcpy(str, expression);
306. }
307. void multiplay2(char *s, char *str){
308.     int x=0;
309.     char expression[100];
310.     strcpy(expression,"");
311.     char eq[100]={""};
312.      strcpy(eq,s);
313.      int i=0;
314.     if(eq[0] != '-'){
315.         i=1;
316.     }
```

Table 3.10: C Language Computer Programming Code

**C Language Computer Programming Code**

```
317.         char *copy=strdup(eq);
318.         char *delim="+-";
319.         char *tok=strtok(eq, delim);
320.         while(tok!=NULL){
321.             char temp[100]={""};
322.             if(x==0){
323.                 strcpy(temp, "-");
324.             }
325.             else{
326.                 strcpy(temp, "+");
327.             }
328.      if(i != 1){
329.   strcat(temp, tok);
330.  strcat(temp , str);
331.  strcat(expression, temp);
332.             }
333.    else{
334.  char *t=strdup(tok);
335. strcat(t, str);
336. strcat(expression, t);
337.             }
338.  int from = tok-eq+strlen(tok);
339.  tok=strtok(NULL,delim);
340.  int to = tok != NULL ? tok-eq : strlen(copy);
341.             char c[4];
342.             sprintf(c, "%.*s", to-from, copy+from);
343.                 if(c[0] == '+'){
344.                     x=1;
345.                 }
346.                 else if(c[0] == '-'){
347.                     x=0;
348.                 }
349.             i--;
350.         }
351.         strcpy(str, expression);
352. }
353. void delete_zeros(char* str){
354.     char *ptr;
355.  while((ptr=strstr(str, "+00")) != NULL ||
        (ptr=strstr(str, "00+")) != NULL
       || (ptr=strstr(str, "-00")) != NULL ){
356.  int i=3;
357.         int j=0;
```

Table 3.11: C Language Computer Programming Code

**C Language Computer Programming Code**

```
358.     while(*(ptr + i) != '\0'){
359.   *(ptr + j)=*(ptr + i);
360.             i++;
361.             j++;
362.         }
363.         *(ptr+j)='\0';
364.     }
365. while((ptr=strstr(str, "00-")) != NULL){
366.   replace_string(str,"00-","-");
367.     }
368. }
369. void row_positions(String mat[x][y], int r,
        String *equations, String *variable, int v){
370.     String row[1][y];
371.     String var[v];
372.  for(int m=0;m<v;m++){
373.         strcpy(var[m].field , variable[m].field);
374.     }
375.  for(int i=0;i<x;i++){
376.  for(int j=0;j<y;j++){
377.    if(i==r){
378.    strcpy(row[0][j].field , mat[i][j].field);
379.      continue;
380.             }
381.         }
382.     }
383.     Rpositioning(row, equations, var, v);
384. }
385. void Rpositioning(String row[1][y], String *equations,
        String *var, int v){
386.    String matrix[x][y];
387.    for(int i=0; i<x; i++){
388.    for(int i=0; i<x; i++){
389.    for(int j=0;j<y; j++){
390.   strcpy(matrix[i][j].field , "00");
391.                 }
392.             }
393.  for(int j=0; j<y; j++){
394. strcpy(matrix[i][j].field , row[0][j].field);
395.         }
396.    printf("\n Row Position:-\n");
397.    display_Matrix(matrix);
398.  formEquation(matrix,equations, var ,v);
399.     }
400. }
```

Table 3.12: C Language Computer Programming Code

**C Language Computer Programming Code**

```
401. void col_positions(String mat[x][y], int c,
        String *equations, String *variable, int v){
402.     String col[x][1];
403.     String var[v];
404.     for(int m=0;m<v;m++){
405.         strcpy(var[m].field , variable[m].field);
406.      }
407.     for(int i=0;i<x;i++){
408.     for(int j=0;j<y;j++){
409.    if(j==c){
410.                 strcpy(col[i][0].field , mat[i][j].field);
411.                 continue;
412.             }
413.         }
414.     }
415.     Cpositioning(col, equations, variable, v);
416. }
417. void Cpositioning(String col[x][1], String *equations,
       String *var, int v){
418.     String matrix[x][y];
419.   for(int i=0; i<y; i++){
420.    for(int i=0; i<x; i++){
421.     for(int j=0;j<y; j++){
422.   strcpy(matrix[i][j].field , "00");
423.                 }
424.             }
425.         for(int j=0; j<x; j++){
426.   strcpy(matrix[j][i].field , col[j][0].field);
427.     }
428.     printf("\n Column Position:-\n");
429.     display_Matrix(matrix);
430. formEquation(matrix,equations, var ,v);
431.     }
432.}
433. void zero_mul(char *str){
434.     char *z;
435.     for(int i=0;str[i]!='\0' ;i++){
436.         if(str[i] == '0'){
437.             int p=i;
438.             int n=i;
439.   while(str[p]!='+' || str[p]!='-' || p!=0){
440.                 p--;
441.                 str[p]='0';
442.             }
```

Table 3.13: C Language Computer Programming Code

**C Language Computer Programming Code**

```
443.              while(str[n]!='+' || str[n]!='-'){
444.                  i=n;
445.                  n++;
446.                  str[n]='0';
447.              }
448.          }
449.      }
450. }
451. bool compareTo(char* str){
452.     char *str2=strdup(str);
453.     char c[6];
454.     char u1[2][6];
455.     strcpy(c,"e");
456.      strcpy(u1[0], "u");
457.     strcpy(u1[1],"u");
458.      int u=0;
459.     for(int i=0;str[i] != '\0' ;i++){
460.         if(str[i] == 'u'){
461.             int j=i+1;
462.     while( isdigit(str[j])){
463.     char cc[]={str[j],'\0'};
464.      strcat(u1[u], cc);
465.                  j++;
466.              }
467.              u++;
468.          }
469.          if(str[i] == 'e'){
470.              int j=i+1;
471.              while(isdigit(str[j]) ){
472.                  char cc[]={str[j],'\0'};
473.                  strcat(c, cc);
474.                  j++;
475.              }
476.          }
477.     }
478.     strcpy(str2 , u1[1]);
479.     strcat(str2, c);
480.     strcat(str2, u1[0]);
481.     for(int i=0;i<col_count;i++){
482.    if(strcmp(cols[i].field , str) == 0
         || strcmp(cols[i].field, str2) == 0){
483.              return false;
484.          }
485.     }
```

Table 3.14: C Language Computer Programming Code

**C Language Computer Programming Code**

```
486.      return true;
487. }
488. char* getsimilarof(char *str){
489.      char *str2=strdup(str);
490.      char c[6];
491.     char u1[2][6];
492.      strcpy(c,"e");
493.      strcpy(u1[0], "u");
494.      strcpy(u1[1],"u");
495.     int u=0;
496.     for(int i=0;str[i] != '\0' ;i++){
497.         if(str[i] == 'u'){
498.     int j=i+1;
499.   while( isdigit(str[j]) ){
500.   char cc[]={str[j],'\0'};
501.    strcat(u1[u], cc);
502.                 j++;
503.             }
504.             u++;
505.         }
506.         if(str[i] == 'e'){
507.             int j=i+1;
508.   while(isdigit(str[j]) ){
509.      char cc[]={str[j],'\0'};
510.     strcat(c, cc);
511.                 j++;
512.             }
513.         }
514.     }
515.     strcpy(str2 , u1[1]);
516.     strcat(str2, c);
517.     strcat(str2, u1[0]);
518.     return str2;
519. }
520. void form_matrix(String* equation,int x){
//getting columns
521.     String eq[x-1];
522.      for(int i=0;i<x-1;i++){
523.   strcpy(eq[i].field,equation[i].field);
524.     }
525.   cols=(String*) malloc(x* 400* sizeof(String));
526.     for(int i=0;i<x-1;i++){
527.   char *tok=strtok(eq[i].field, "+-");
528.    while(tok != NULL){
```

Table 3.15: C Language Computer Programming Code

**C Language Computer Programming Code**

```
529.      if(compareTo(tok)){
530.    strcpy(cols[col_count].field, tok);
531.     col_count++;
532.             }
533.    tok=strtok(NULL,"+-");
534.         }
535.      }
536. printf("\n\n");
537. float matrix[x-1][col_count];// final matrix creation
538. printf("\n Matrix order:[%d * %d] \n",x-1,col_count);
//Initialization of final matrix
539.  for(int i=0; i< x-1; i++){
540.   for(int j=0; j< col_count ;j++){
541.   matrix[i][j]=0;
542.         }
543.    }
544.   for(int j=0;j< col_count ; j++){
545.    char *str2=getsimilarof(cols[j].field);
546.   for(int i=0;i< x-1; i++){
547.             char* ptr;
548.   if(((ptr=strstr(equation[i].field,str2)) != NULL)
        || (ptr=strstr(equation[i].field,cols[j].field))!=NULL){
549.   if(ptr == &(equation[i].field[j])){
550.   matrix[i][j]=1;
551.   continue;
552.                     }
553.  else if(*(ptr -1)=='-' ){
554.            matrix[i][j]=-1;
555.                     }
556.                     else{
557.  matrix[i][j]=1;
558.    continue;
559.                     }
560.                }
561.             }
562.         }
563.    printf("\nFinally the Matrix is:-\n\n");
564.    display_2D_array(x-1, col_count, matrix);
565. printf("\n*******Now; The Reduced Row Echelon Form is:********\n");
566. rref(x-1, col_count, matrix);
567. }
568. void display_2D_array(int x, int y,float matrix[][y]){
569.     printf("\n  \t");
570.    float f=0;
```

Table 3.16: C Language Computer Programming Code

**C Language Computer Programming Code**

```
571.     for(int i=0; i< y; i++){
572.         printf("| %10s",cols[i].field);
573.     }
574.     for(int i=0;i<x;i++){
575.   printf("\neq.%3d :",i+1);
576.   for(int j=0;j<y;j++){
577.    if(matrix[i][j] == 0){
578.    printf("| %10.0f",f);
579.                 }
580.                 else{
581.    printf("| %10.0f",matrix[i][j]);
582.                 }
583.         }
584.     }
585. }
586. void normalize(int x1,int y1, int x,
         int y , float matrix[][y]){
587.     for(int i=x1+1;i<x;i++){
588.      if(matrix[i][y1] != 0 ){
589.       float t=matrix[i][y1];
590.        for(int j=0;j<y;j++){
591.    matrix[i][j]=matrix[i][j] + ((-1)* t * matrix[x1][j]);
592.                   }
593.           }
594.     }
595.    for(int i=x1-1;i>=0;i--){
596.             if(matrix[i][y1] != 0){
597.                 float t=matrix[i][y1];
598.                 for(int j=0;j<y;j++){
599.     matrix[i][j]=matrix[i][j] + ((-1)* t * matrix[x1][j]);
600.             }
601.          }
602.      }
603. }
604.   void swapeRows(int x1,int x2,int y,
605.           float matrix[][y]){
606.     for(int j=0;j<y;j++){
607.   int temp=matrix[x1][j];
608. matrix[x1][j]=matrix[x2][j];
609.         matrix[x2][j]=temp;
610.     }
611. }
```

Table 3.17: C Language Computer Programming Code

**C Language Computer Programming Code**

```
612. void rref(int x,int y, float matrix[][y]){
//display_2D_array(x, y, matrix);
613. int i=0;
614.     int j=0;
615. while(i<x && j<y){
616.   if(matrix[i][j]!=0){
617.   float tem=matrix[i][j];
618.   for(int m=0;m<y;m++){
619.   matrix[i][m]=matrix[i][m] / tem;
620.             }
621.    normalize(i,j,x,y,matrix);
622.         }
623.    else if(matrix[i][j] == 0){
624.    int k=0;
625.   for(int m=i+1;m<x;m++){
626.   if(matrix[m][j] != 0){
627.    k=1;
628.  float tem=matrix[m][j];
629.    for(int n=0;n<y;n++){
630.    matrix[m][n]= (matrix[m][n] / tem);
631.  }
632.   swapeRows(i,m,y,matrix);
633.    break;
634.          }
635.            }
636.        if(k==0){
637.    i--;
638.        }
639.        else{
640. normalize(i,j,x,y,matrix);
641.        }
642.      }
643.    i++;
644.    j++;
645.    }
646.  display_2D_array(x, col_count, matrix);
647.       printf("\n");
648.  printf("\n\n\n*****FINAL RESULT*****\n");
649.  int check=EIDS_Check(x,col_count , matrix);
650.    if( check == 0){
651.        printf("\n******This is EIDS ******\n");
652.    }
```

Table 3.18: C Language Computer Programming Code

**C Language Computer Programming Code**

```
653.     else{
654.   printf("\n******This is not EIDS ******\n");
655.     }
656. }
657. int EIDS_Check(int x, int y, float matrix[][y]){
658. for(int i=0; i< x ;i++){
659.         for(int j=i;j<y;j++){
660.    int no=(int) matrix[i][j];
661.   if(no != 0){
662.   for(int m=0;m<y;m++){
663.   if((j != m) && (i!=m) && (matrix[i][m] == 1)){
664. printf("\n%d  %d= %d\n",i,j,no);
665.   return 1;
666.                           }
667.                       }
668.                   }
669.         }
670.     }
671.     return 0;
672. }
673. int main(){
674.      int v;
//Inputs
675.     printf("\nEnter the no. of rows:");
676.     scanf("%d",&x);
677.     printf("\nEnter the no. of column:");
678.     scanf("%d",&y);
679.     printf("\nNow,Enter the no. of Variables:");
680.     scanf("%d",&v);
681. int no_of_equations=(v*v*v)+(x*x*v)+(y*y*v);
682.     String equations[(v*v*v*v)+(x*x*v)+(y*y*v)];
683.   printf("\nMaximum No. of Equations = %d\n",(v*v)+(x*x*v)+(y*y*v));
684.     String matrix[x][y];
685.     printf("\n\nEntries to matrix:-");
686.     for(int i=0;i<x;i++){
687.         for(int j=0;j<y;j++){
688.   printf("\nEnter A[%d*%d] Entry:",i+1,j+1);
689.   scanf("%s",matrix[i][j].field);
690.         }
691.     }
692.   display_Matrix(matrix);
693. String variables[v];
694.      for(int i=0;i<v;i++){
```

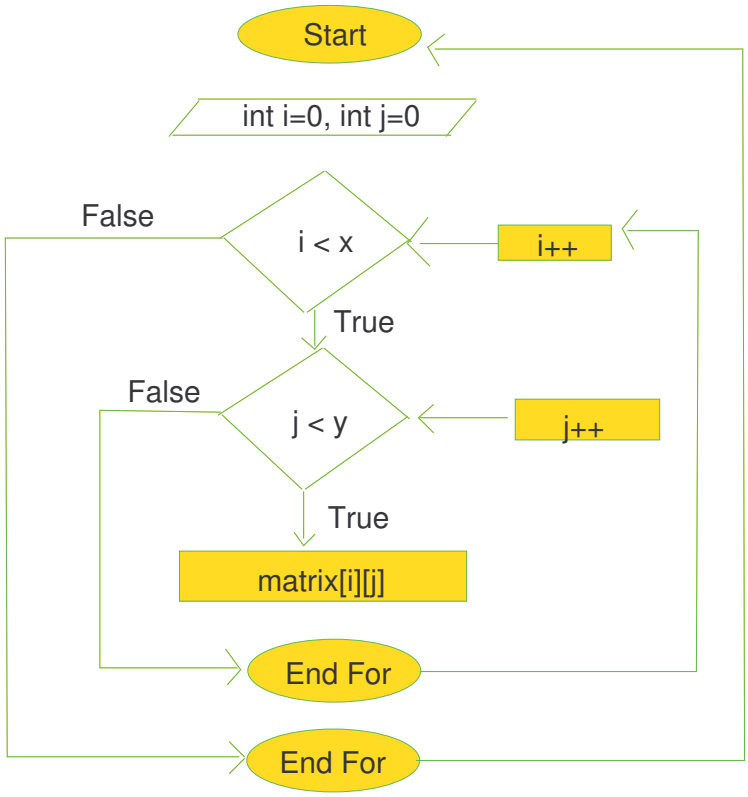
Table 3.19: C Language Computer Programming Code

**C Language Computer Programming Code**

```
695. char temp[5]="u";
696.  char no[4];
697.  itoa(i+1, no,10);
698.   strcat(temp, no);
699.   strcpy(variables[i].field , temp);
700.    }
 //Taking Derivatives
701.    for(int i=0;i<v;i++){
702.  String mat[x][y];
703.  String mat2[x][y];
704.   copy_matrix(matrix, mat);
705.   copy_matrix(matrix, mat2);
706.  derivative2(mat2, variables[i].field , variables, v);
707.  printf("\n\n*****Derivative w.r.t '%s':-\n",variables[i].field);
708.  display_Matrix(mat2);
709.  derivative3(mat, variables[i].field , variables, v);
710.       formDerivateEq(mat, equations,variables,v);
711.       free(mat2);
712.       free(mat);
713.    }
 //Row Positioning
714.  printf("\n********ROW POSITIONS********\n");
715.  for(int i=0;i<x; i++){
716.  String mat[x][y];
717. copy_matrix(matrix, mat);
718.  row_positions(mat, i, equations , variables, v);
719.       free(mat);
720.    }
//Column Positioning
721. printf("\n**********COLUMN POSITIONS*********\n");
722.    for(int i=0;i<y;i++){
723.  String mat[x][y];
724.  copy_matrix(matrix, mat);
725.  col_positions(mat, i, equations , variables, v);
726. free(mat);
727.    }
//EQUATIONS
728.  printf("\n**********EQUATIONS*********\n");
729.    for(int i=0;i<eq_count;i++){
730.       printf("\n %s\n",equations[i].field);
731.    }
732.    printf("\n\n\n\nNOW:-\n");
733.    form_matrix(equations,no_of_equations+1);
734.    printf("\n\n\nPress Any Key to Continue...\n");
735.    getche("");
736.}
```

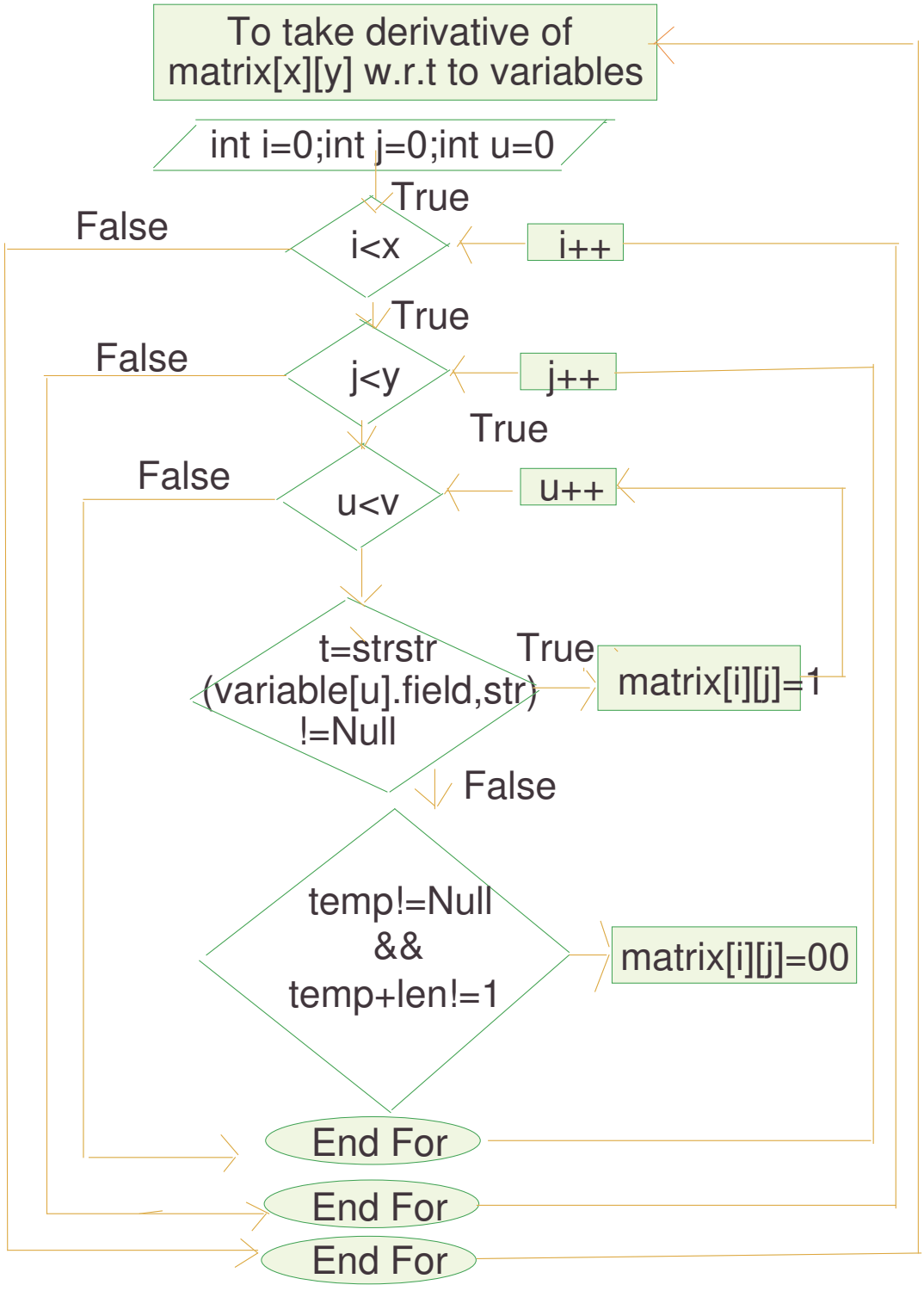
**Now, we construct flow charts to understand the above programme.** We split programme in following steps:

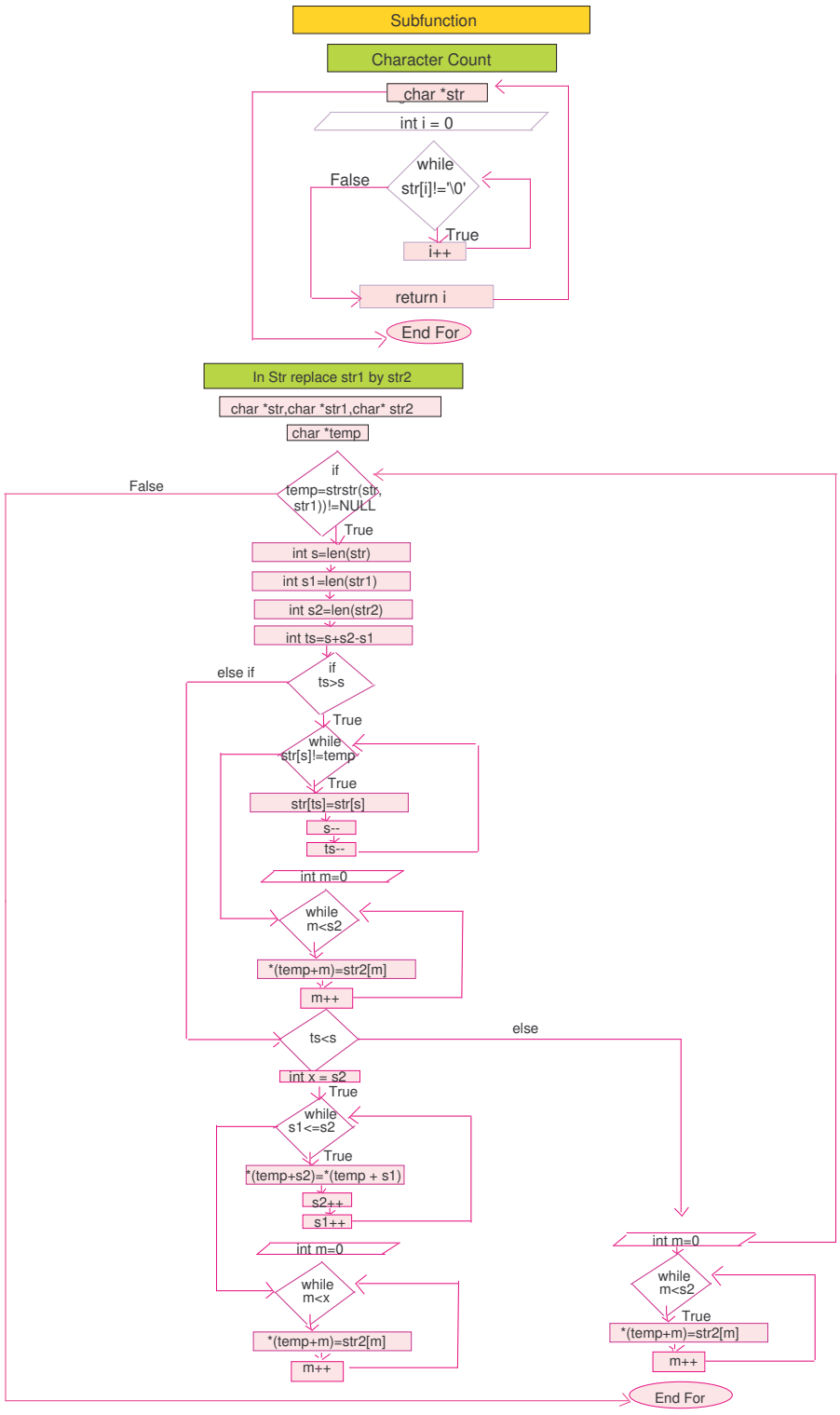
**Step 1:** Formation of Input Matrix



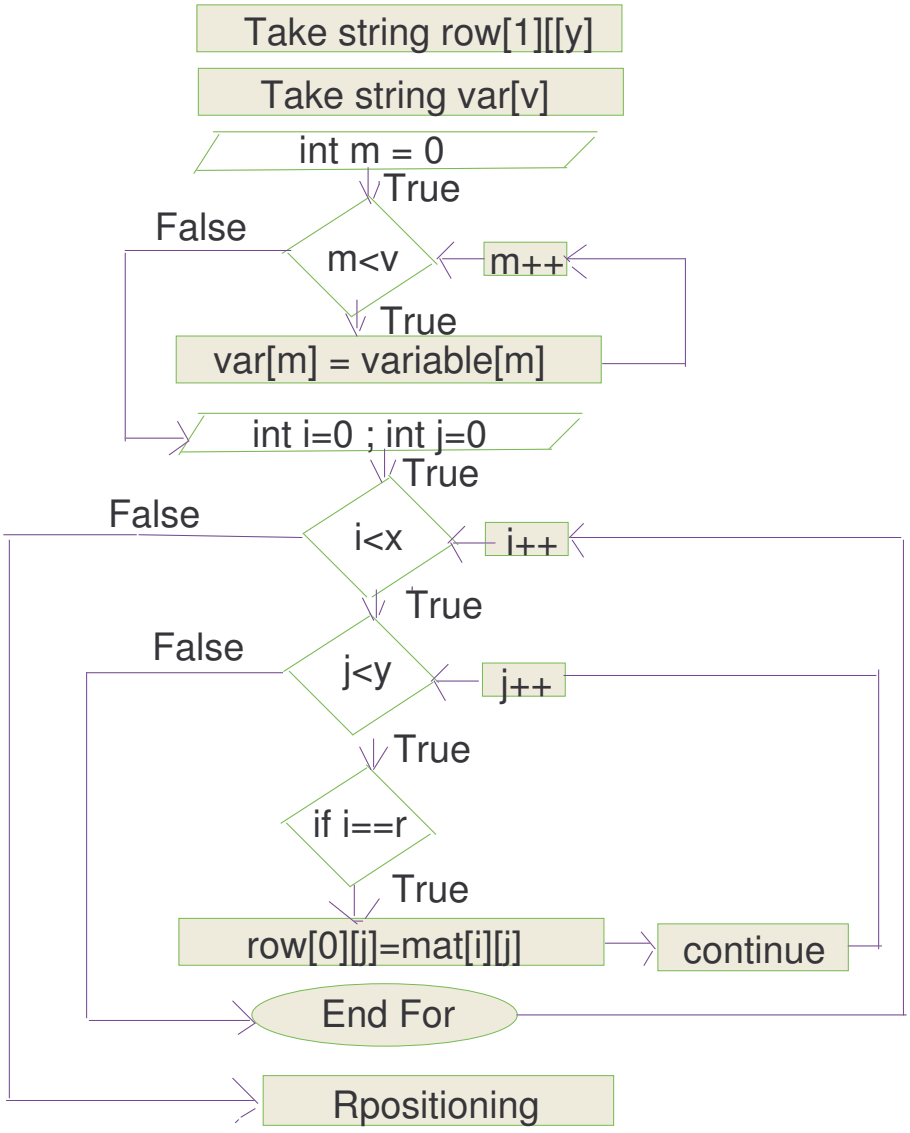
**Step 2:** Generators associated to Jacobian

x=rows
y=columns
v=variables



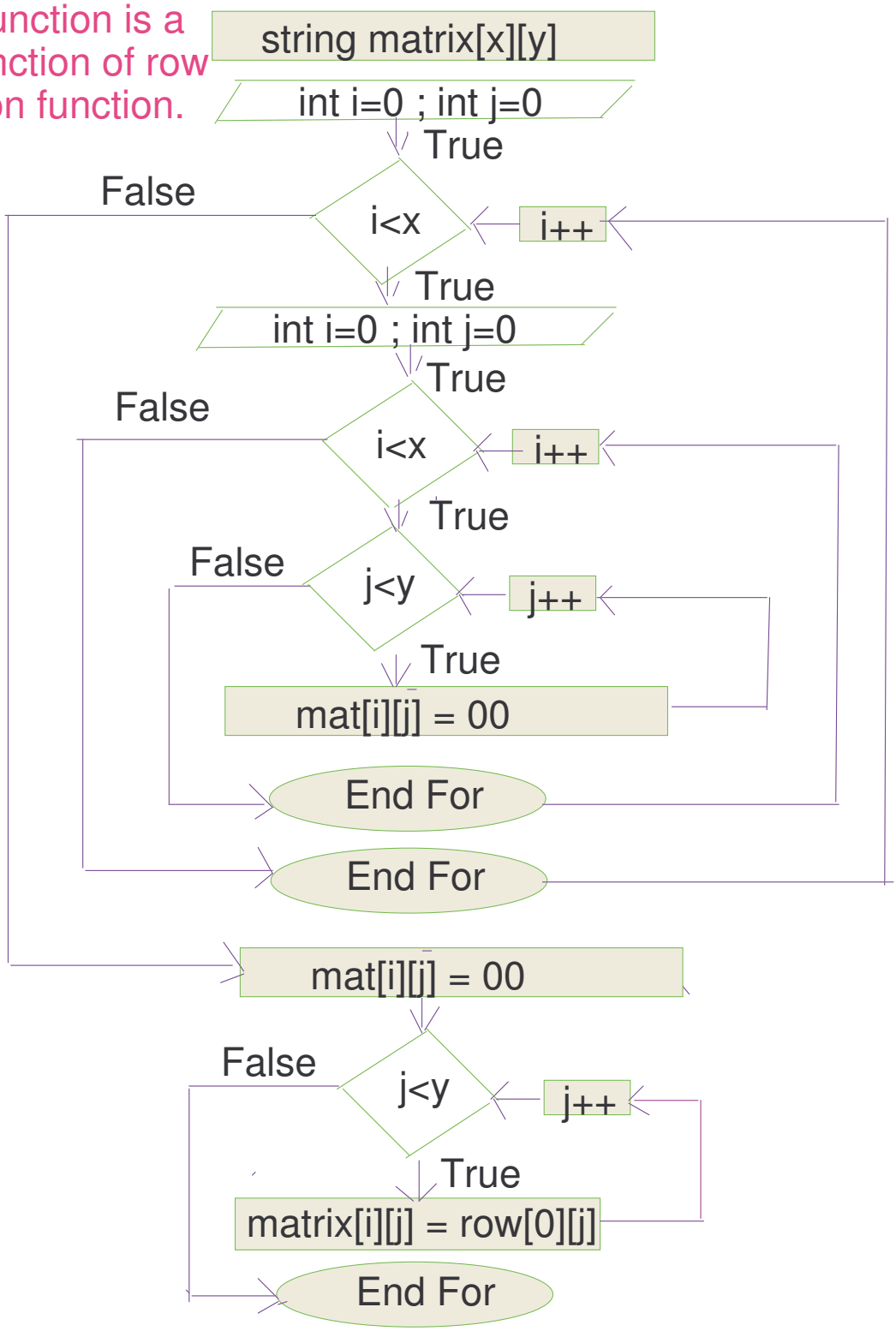

Subfunction
Character Count
char *str
int i = 0
while str[i]!='\0'
False
True
i++
return i
End For
In Str replace str1 by str2
char *str,char *str1,char* str2
char *temp
if temp=strstr(str, str1))!=NULL
False
True
int s=len(str)
int s1=len(str1)
int s2=len(str2)
int ts=s+s2-s1
if ts>s
else if
True
while str[s]!=temp
True
str[ts]=str[s]
s--
ts--
int m=0
while m<s2
*(temp+m)=str2[m]
m++
ts<s
else
int x = s2
True
while s1<=s2
True
*(temp+s2)=*(temp + s1)
s2++
s1++
int m=0
while m<x
*(temp+m)=str2[m]
m++
int m=0
while m<s2
True
*(temp+m)=str2[m]
m++
End For

**Step 3:** Generators associated to Row Conditions

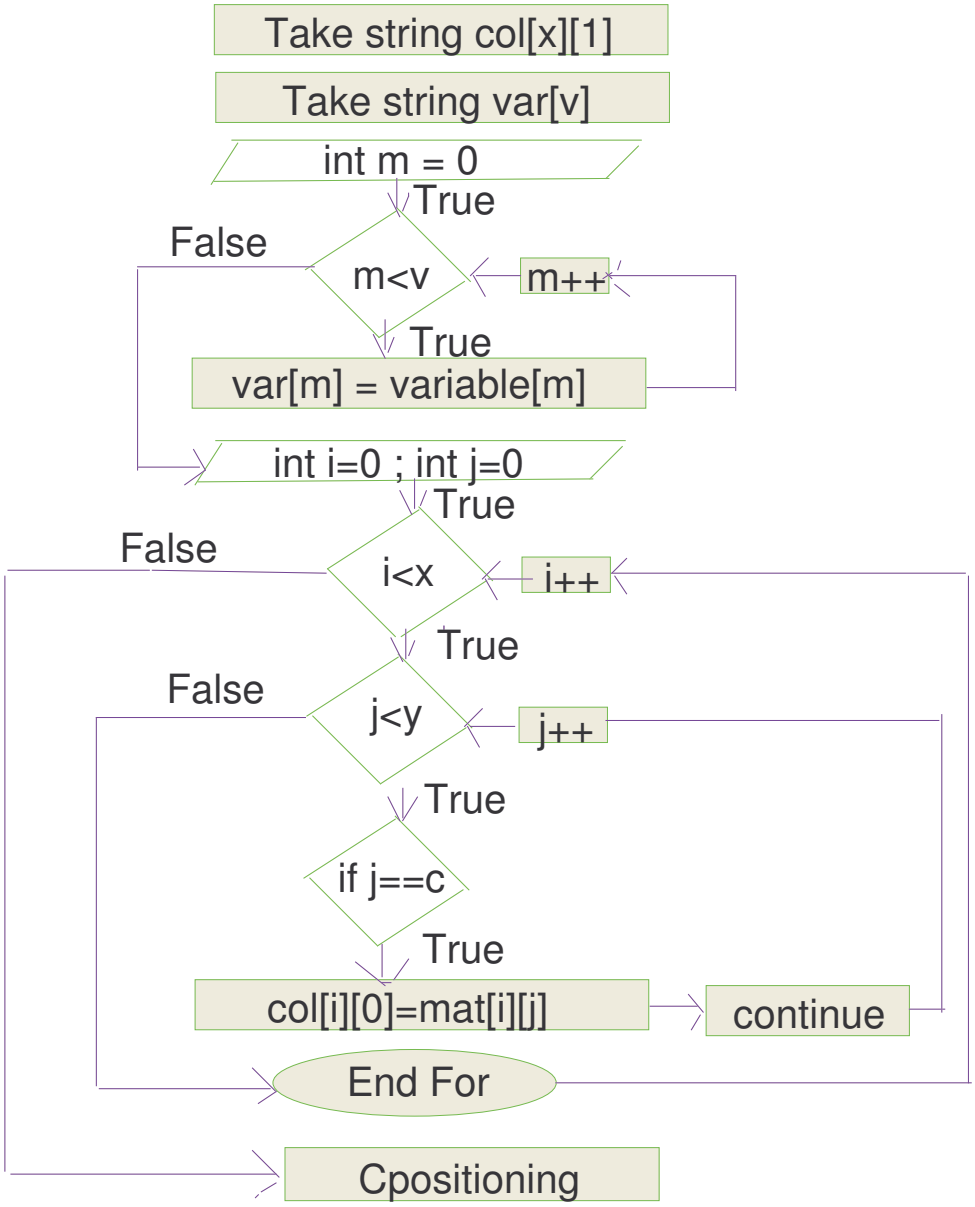


## RPositioning Function

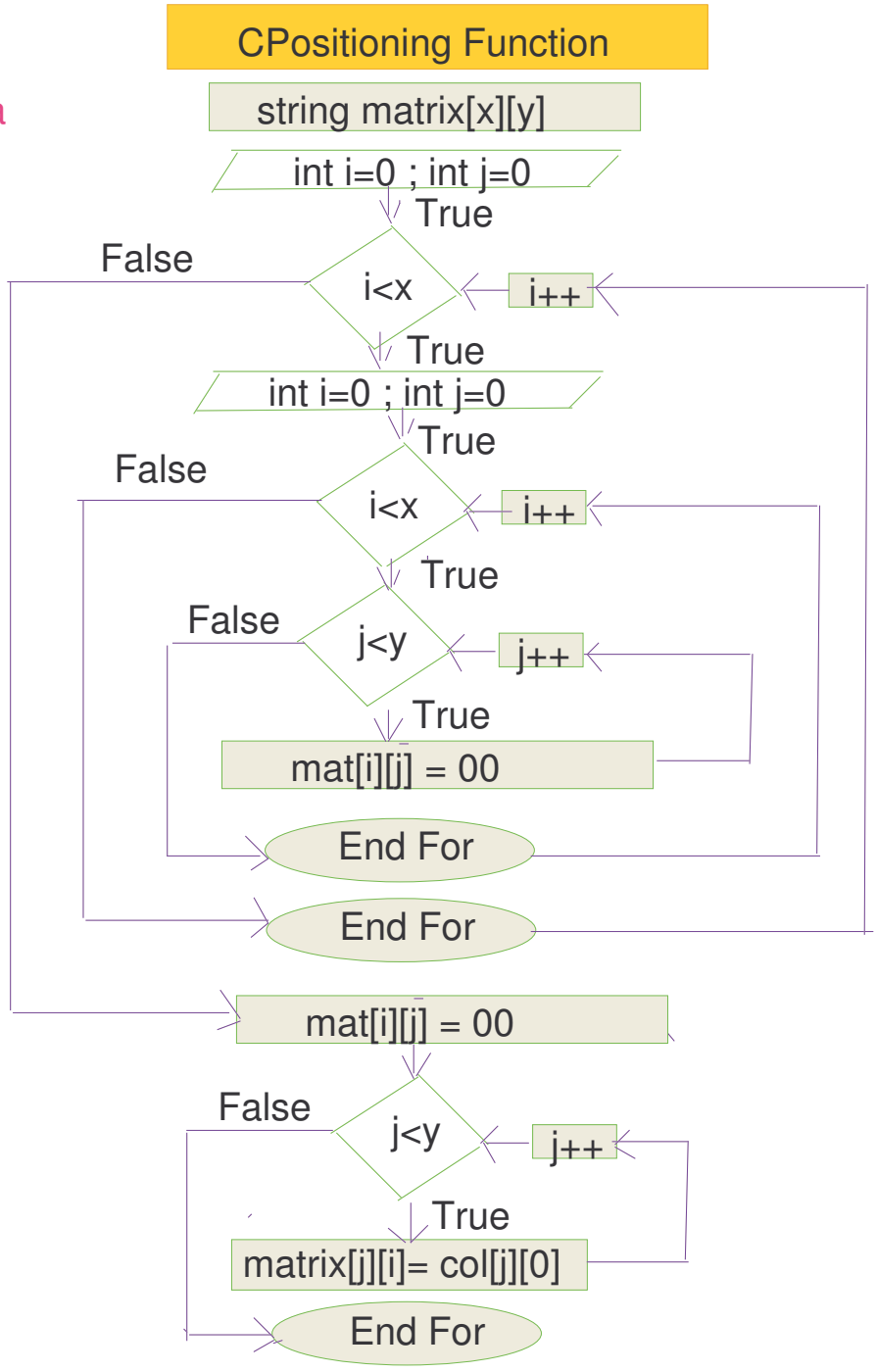
This function is a subfunction of row position function.



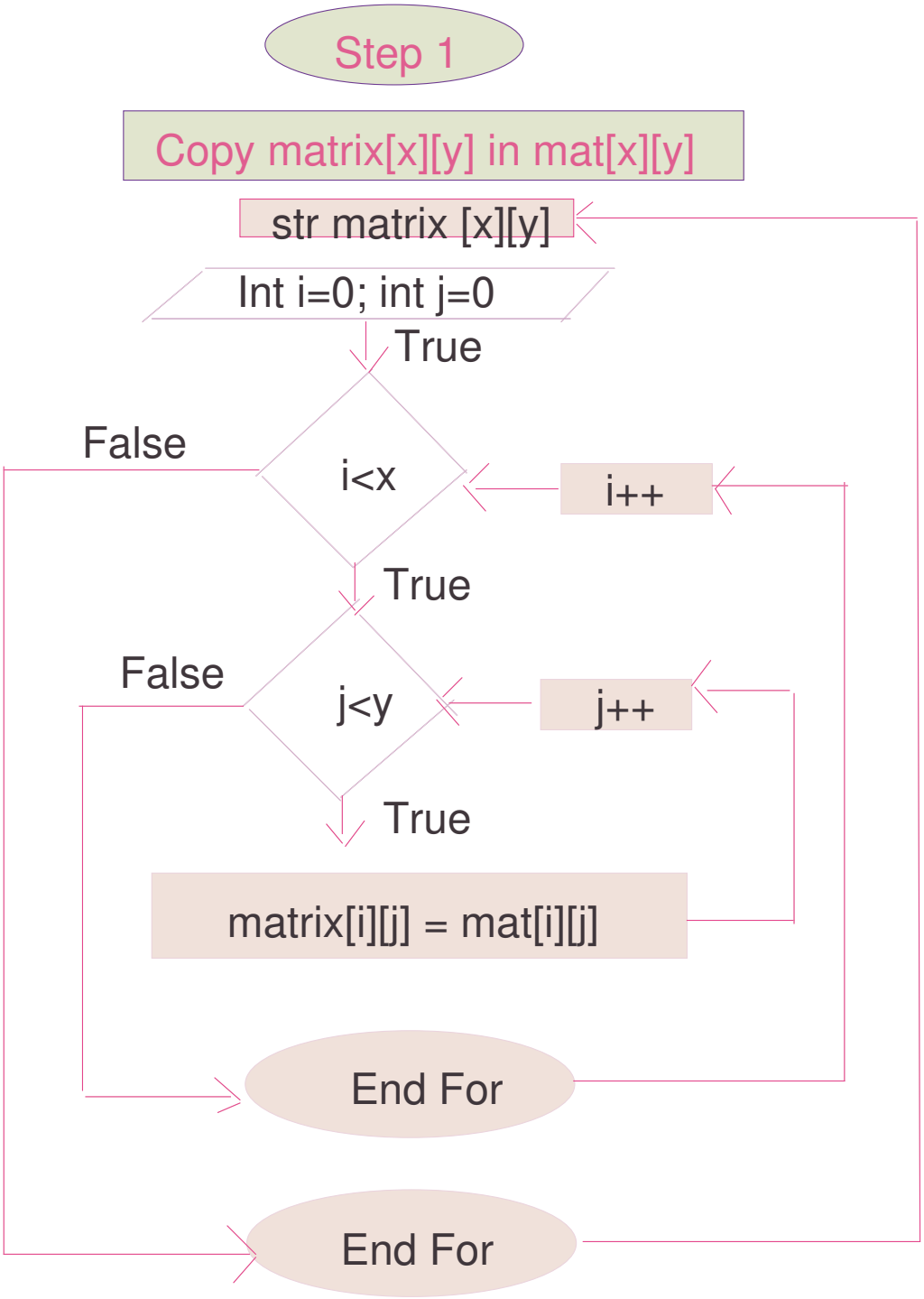
**Step 4:** Generators associated to Column Conditions

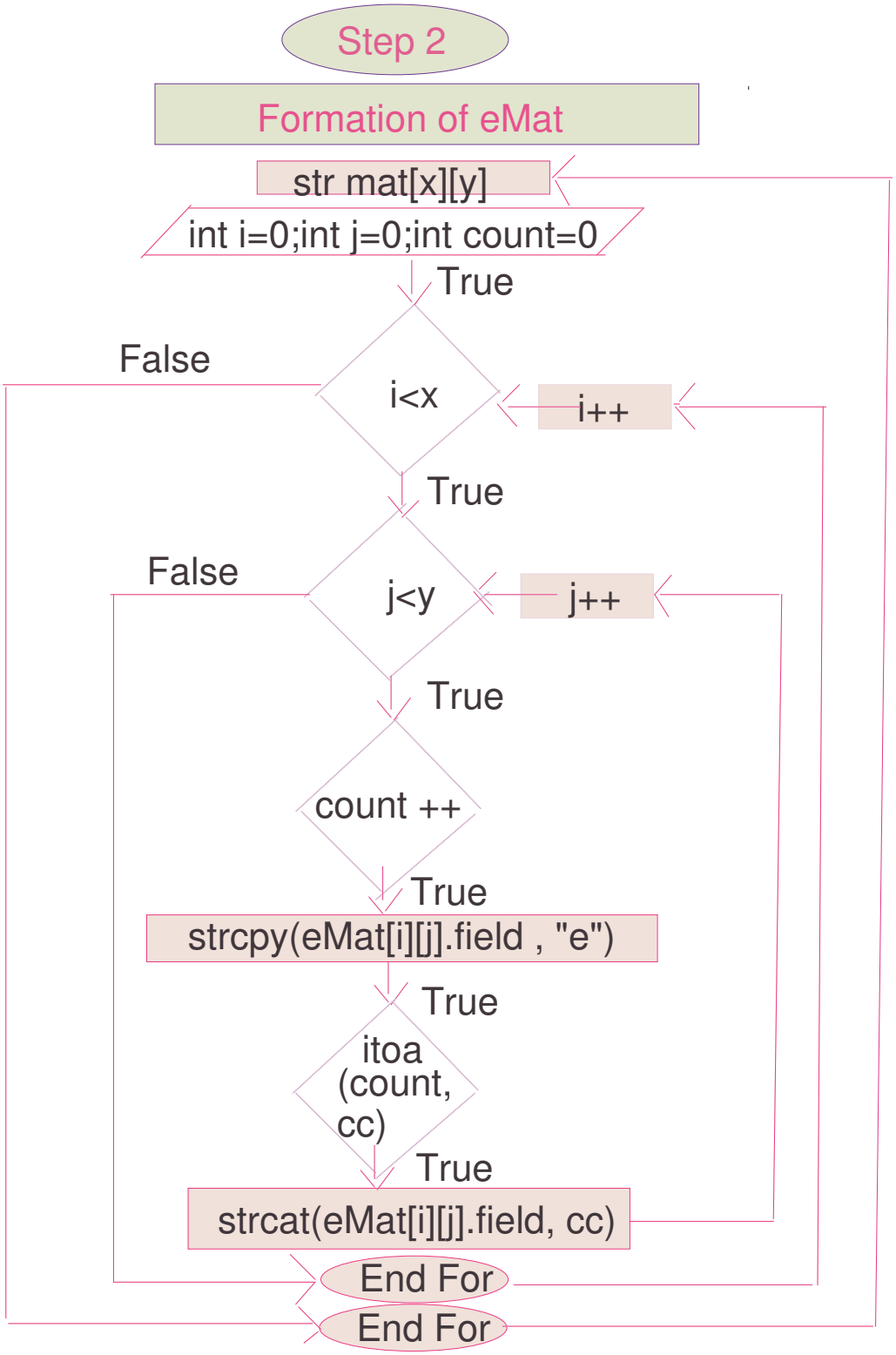


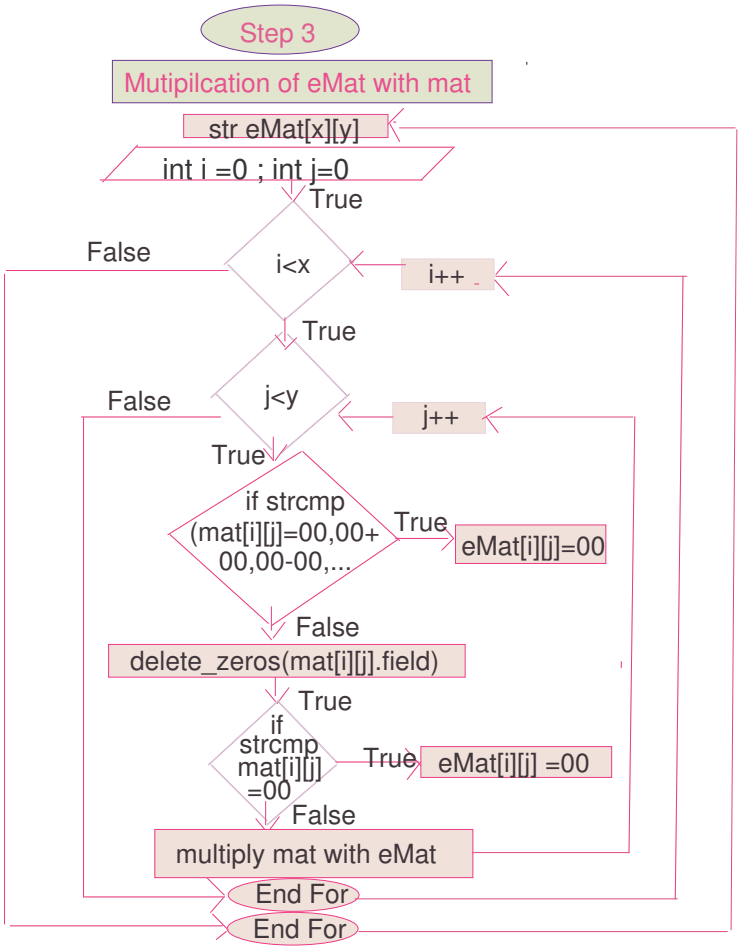
This function is a subfunction of column position function.

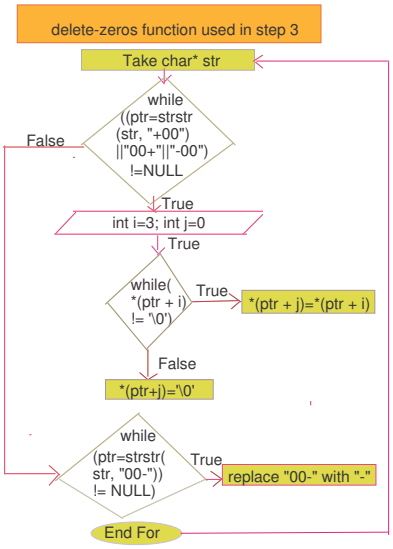


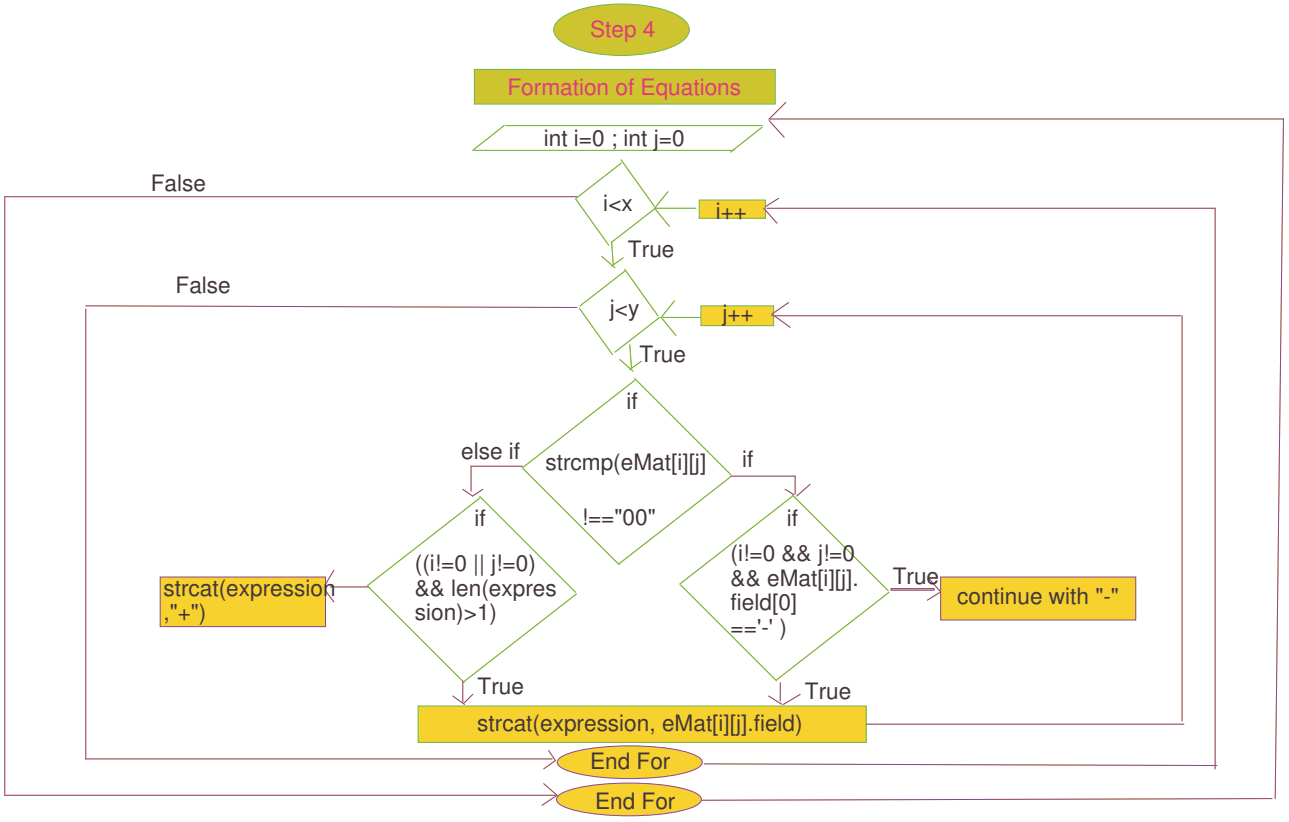
**Step 5:** Formation of Equations

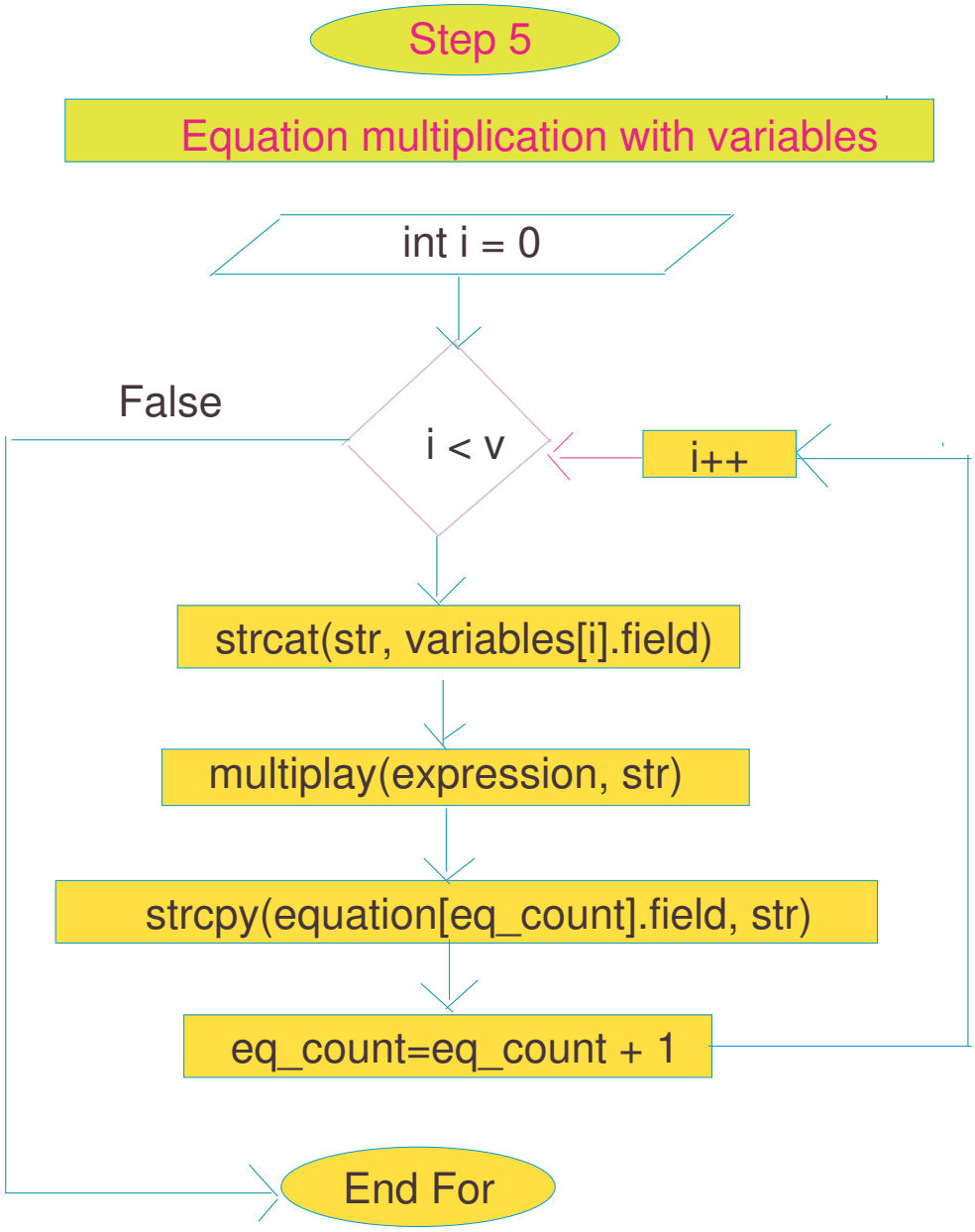


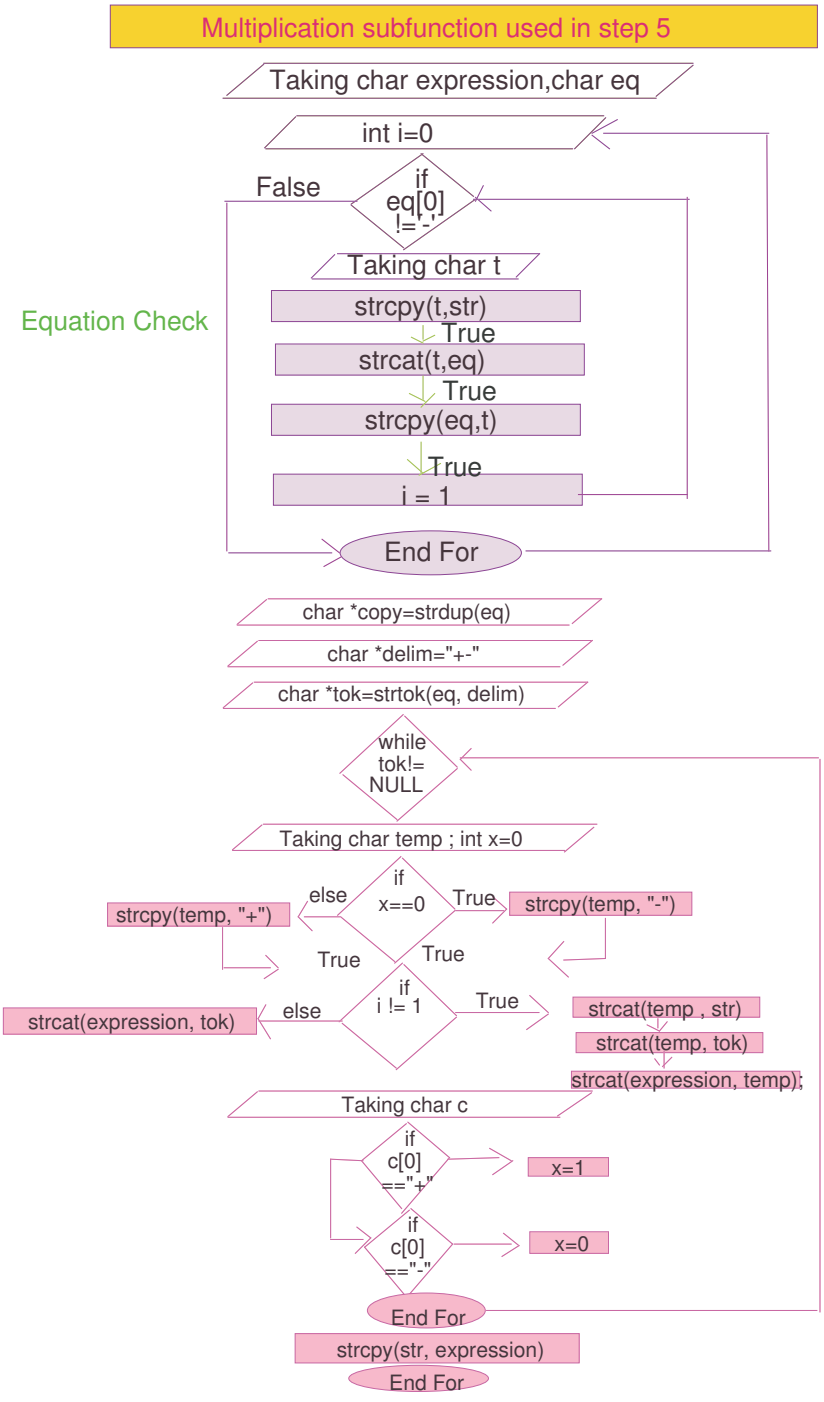

Step 2
Formation of eMat
str mat[x][y]
int i=0;int j=0;int count=0
True
False
i<x
i++
True
False
j<y
j++
True
count ++
True
strcpy(eMat[i][j].field , "e")
True
itoa (count, cc)
True
strcat(eMat[i][j].field, cc)
End For
End For


Step 3
Mutipilcation of eMat with mat
str eMat[x][y]
int i =0 ; int j=0
True
i<x
False
i++
True
j<y
False
j++
True
if strcmp (mat[i][j]=00,00+ 00,00-00,...
True
eMat[i][j]=00
False
delete_zeros(mat[i][j].field)
True
if strcmp mat[i][j] =00
True
eMat[i][j] =00
False
multiply mat with eMat
End For
End For

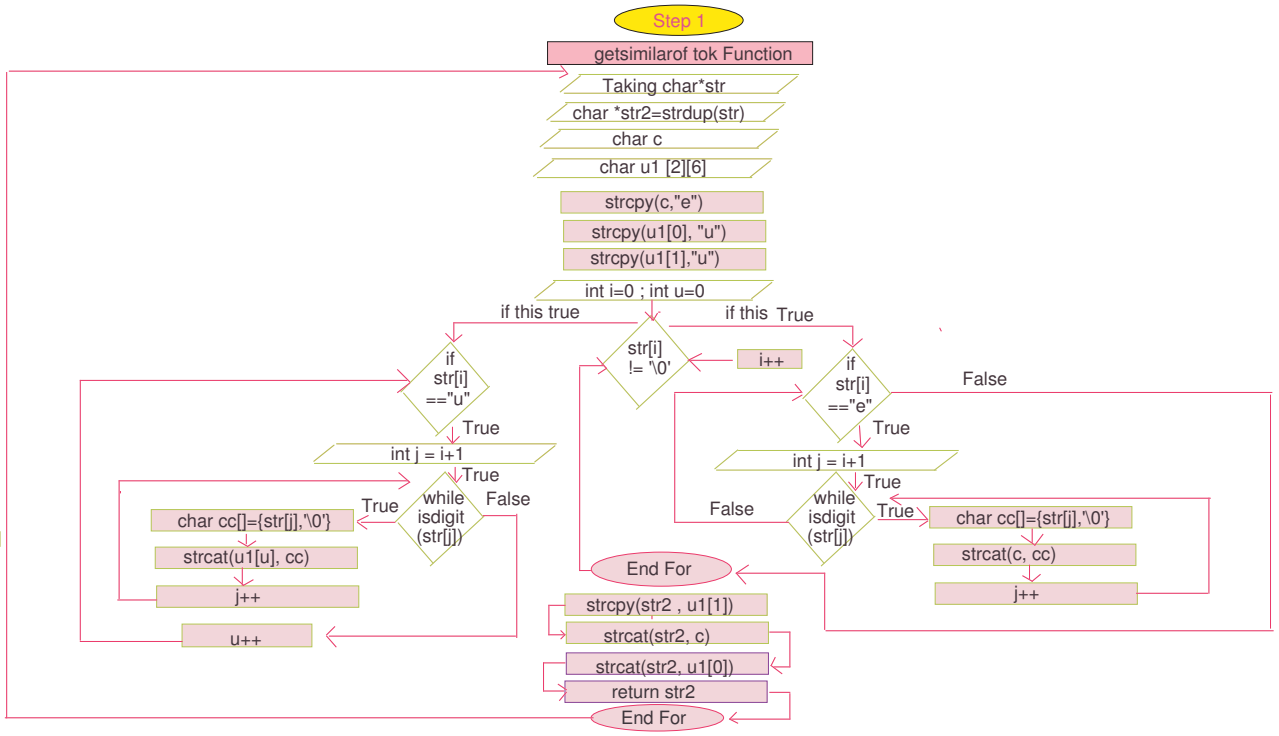

delete-zeros function used in step 3
Take char* str
while ((ptr=strstr (str, "+00") ||"00+"||"-00") !=NULL
False
True
int i=3; int j=0
True
while( *(ptr + i) != '\0')
True
*(ptr + j)=*(ptr + i)
False
*(ptr+j)='\0'
while (ptr=strstr( str, "00-")) != NULL)
True
replace "00-" with "-"
End For

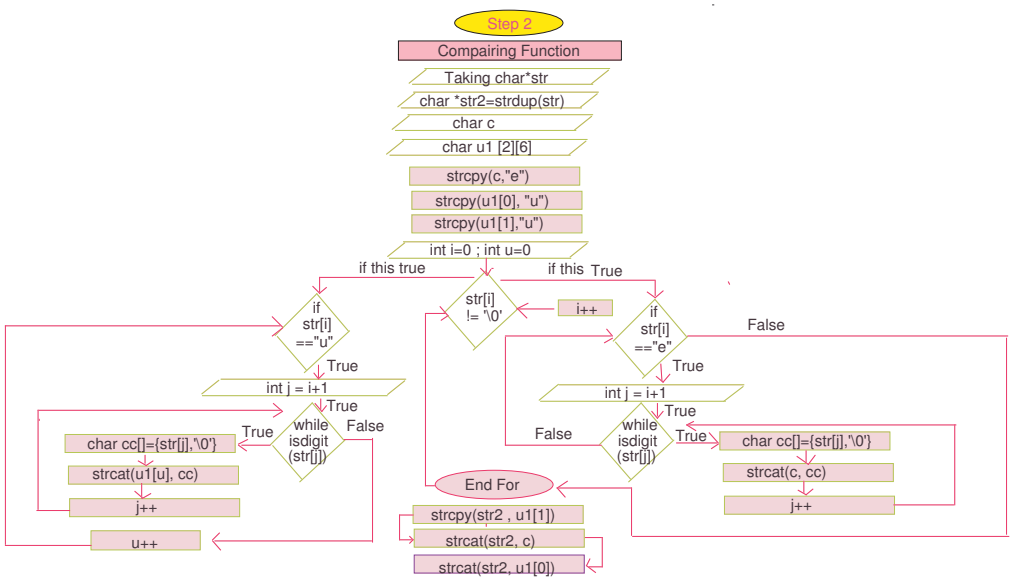

Step 4
Formation of Equations
int i=0 ; int j=0
False
i<x
i++
True
False
j<y
j++
True
if
strcmp(eMat[i][j]
!=="00"
else if
if
((i!=0 || j!=0)
&& len(expres
sion)>1)
strcat(expression
,"+")
True
if
(i!=0 && j!=0
&& eMat[i][j].
field[0]
=='-' )
True
continue with "-"
True
strcat(expression, eMat[i][j].field)
End For
End For

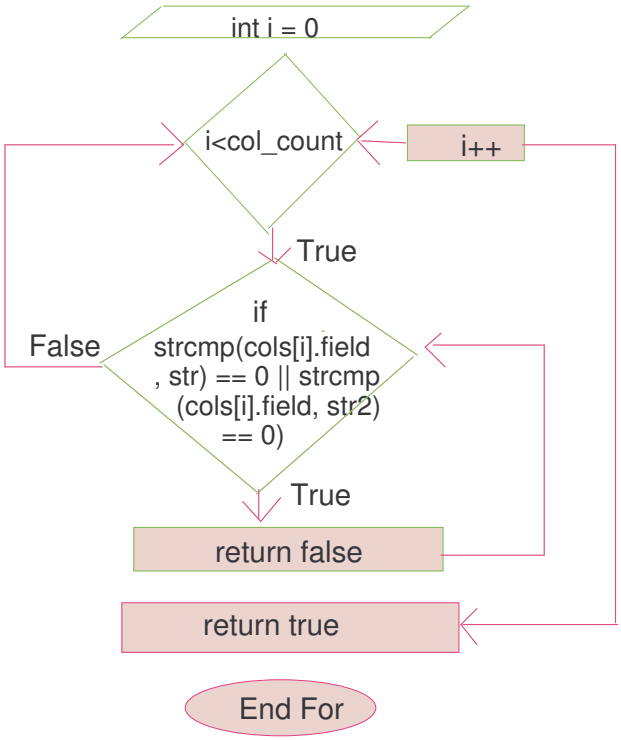

Step 5
Equation multiplication with variables
int i = 0
False
i < v
i++
strcat(str, variables[i].field)
multiplay(expression, str)
strcpy(equation[eq_count].field, str)
eq_count=eq_count + 1
End For

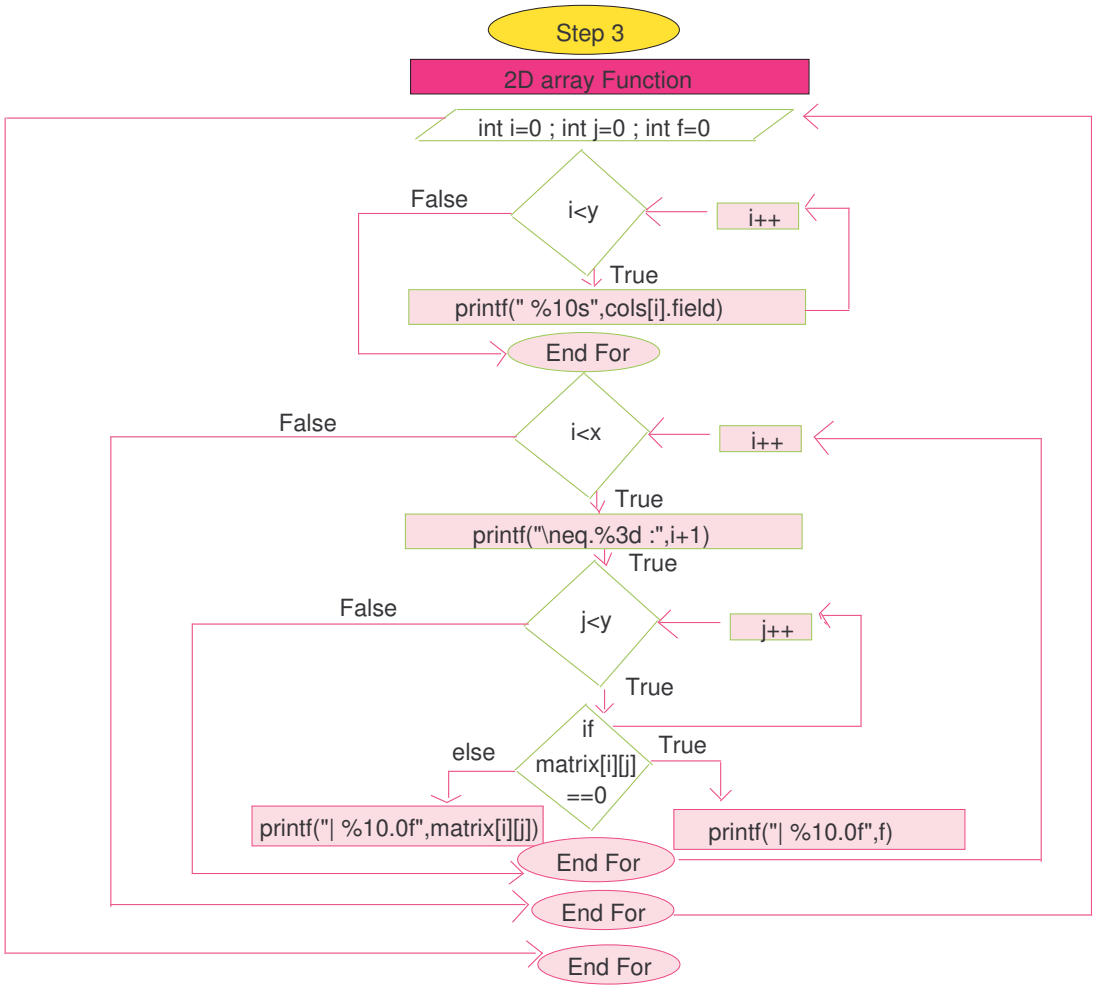

Multiplication subfunction used in step 5
Taking char expression,char eq
int i=0
if eq[0] != '-'
False
Taking char t
Equation Check
strcpy(t,str)
True
strcat(t,eq)
True
strcpy(eq,t)
True
i = 1
End For
char *copy=strdup(eq)
char *delim="+-"
char *tok=strtok(eq, delim)
while tok!= NULL
Taking char temp ; int x=0
if x==0
else
True
strcpy(temp, "+")
strcpy(temp, "-")
True
True
if i != 1
else
True
strcat(expression, tok)
strcat(temp , str)
strcat(temp, tok)
strcat(expression, temp);
Taking char c
if c[0] =="+"
x=1
if c[0] =="-"
x=0
End For
strcpy(str, expression)
End For

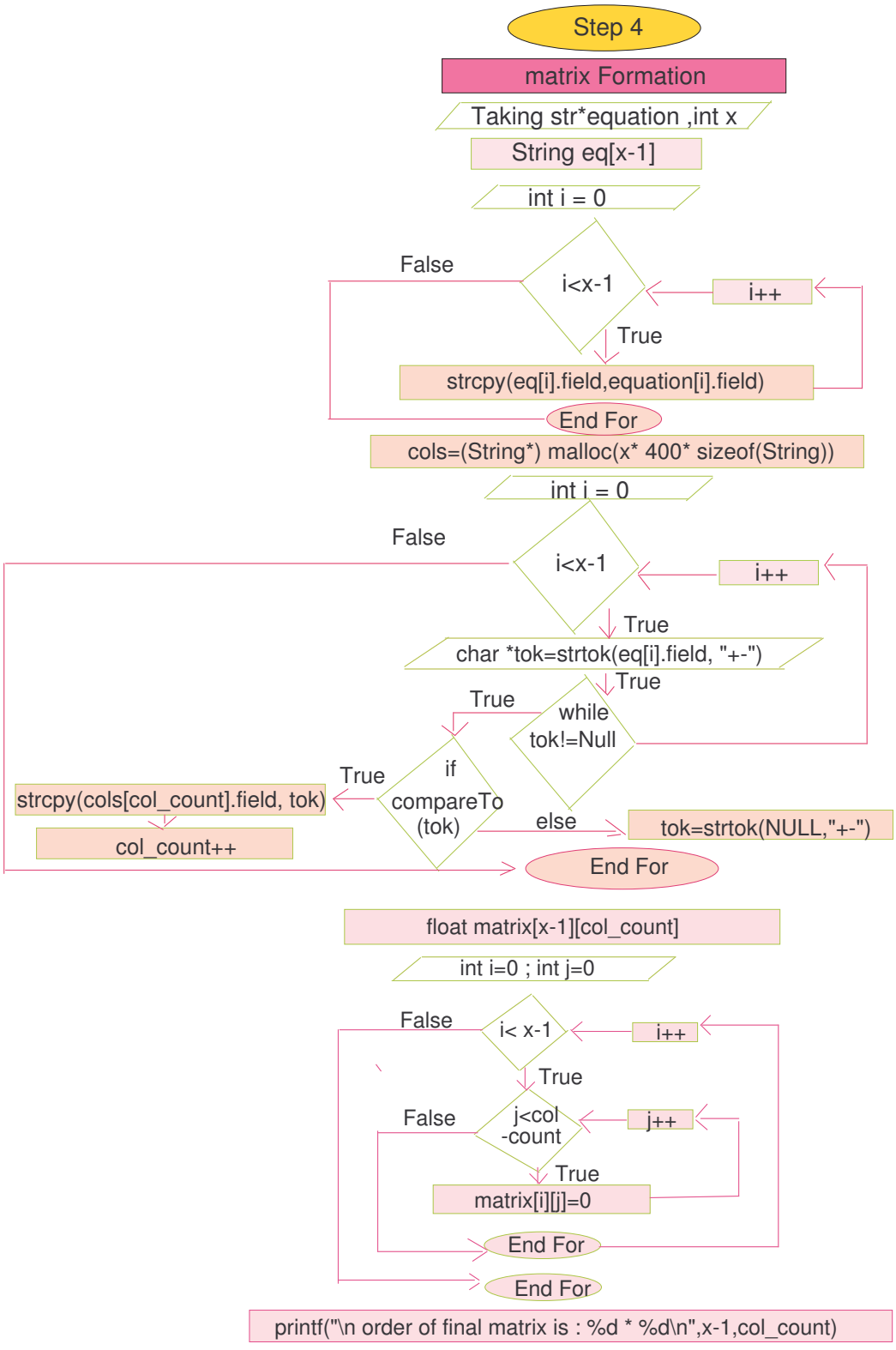
**Step 6:** Formation of Matrix of Co-efficients

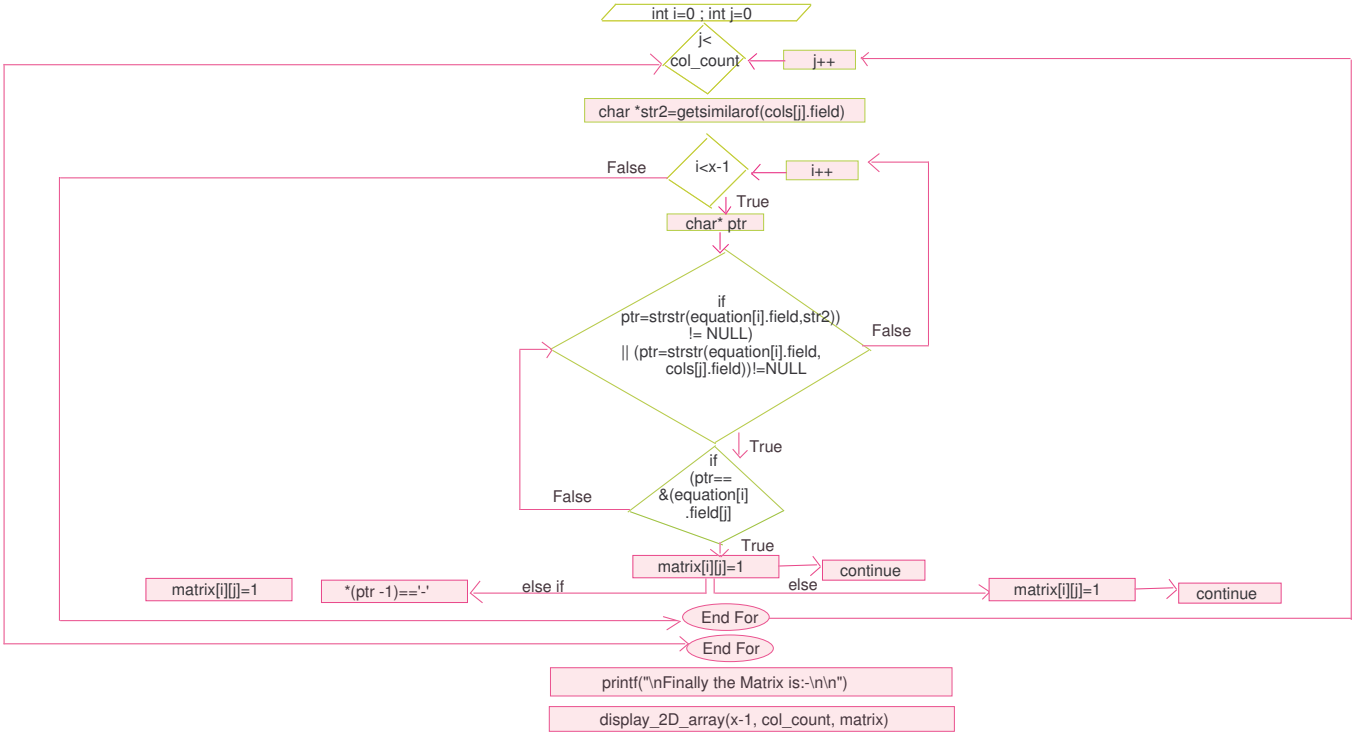



Step 2
Compairing Function
Taking char*str
char *str2=strdup(str)
char c
char u1 [2][6]
strcpy(c,"e")
strcpy(u1[0], "u")
strcpy(u1[1],"u")
int i=0 ; int u=0
if this true
if this True
if str[i] =="u"
str[i] != '\0'
i++
if str[i] =="e"
False
True
int j = i+1
True
while isdigit (str[j])
True
False
char cc[]={str[j],'\0'}
strcat(u1[u], cc)
j++
u++
False
True
char cc[]={str[j],'\0'}
strcat(c, cc)
j++
End For
strcpy(str2 , u1[1])
strcat(str2, c)
strcat(str2, u1[0])

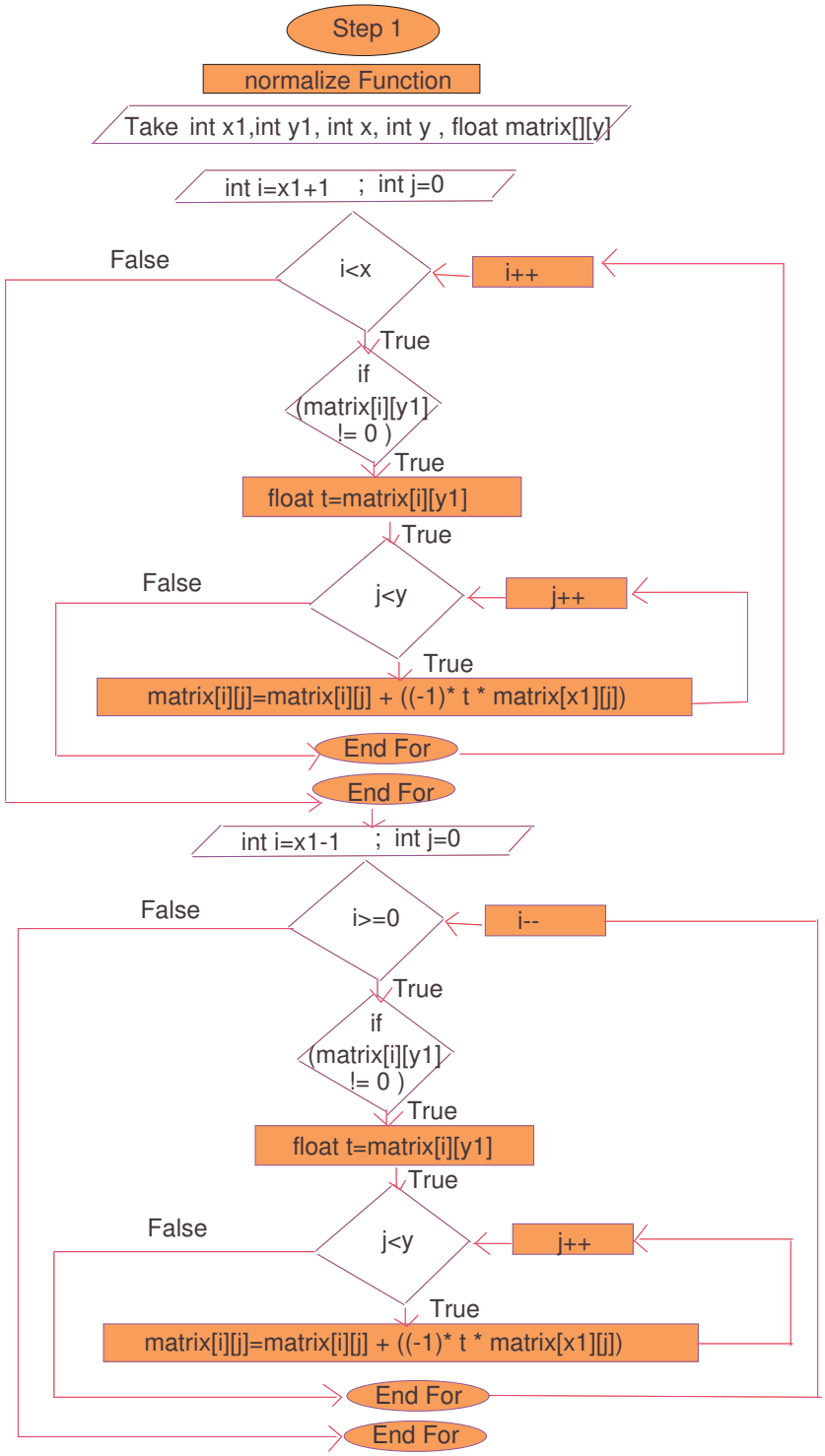

int i = 0
i<col_count
i++
True
if strcmp(cols[i].field , str) == 0 || strcmp (cols[i].field, str2) == 0)
False
True
return false
return true
End For

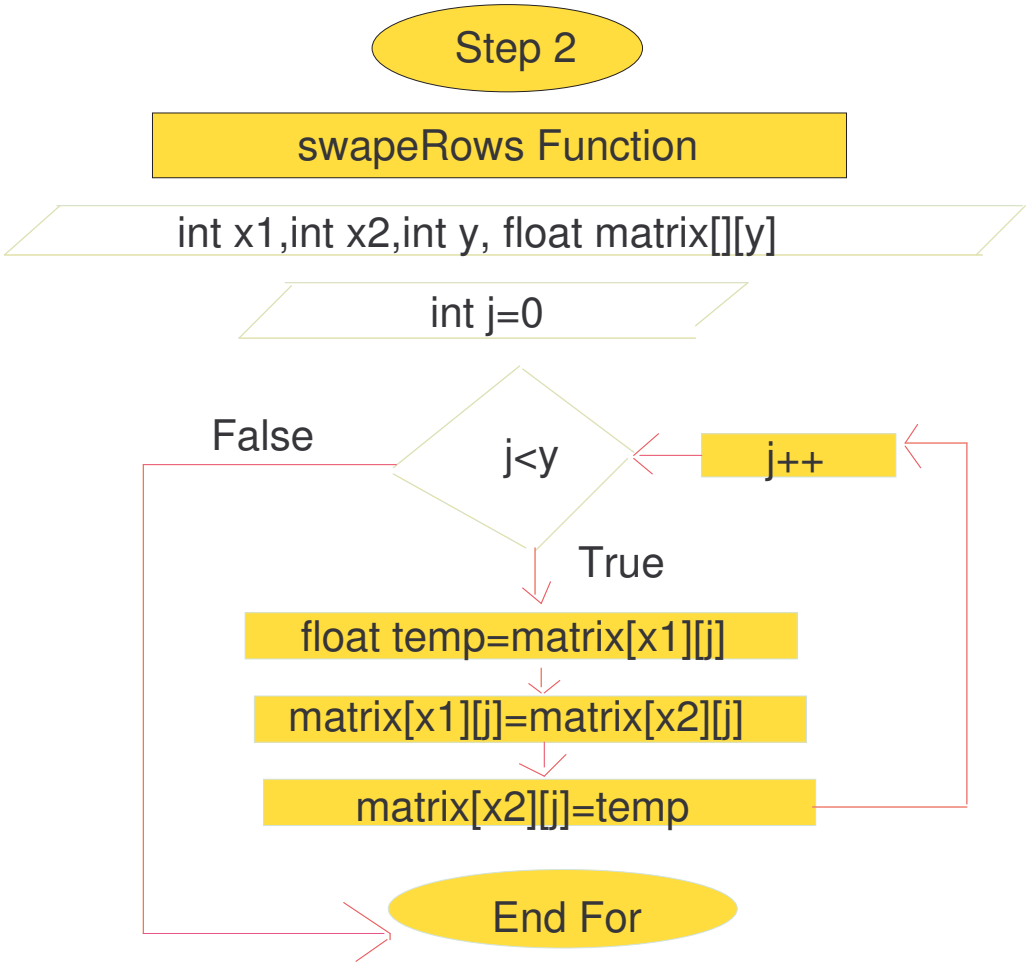

Step 3
2D array Function
int i=0 ; int j=0 ; int f=0
False
i<y
i++
True
printf(" %10s",cols[i].field)
End For
False
i<x
i++
True
printf("\neq.%3d :",i+1)
True
False
j<y
j++
True
if
matrix[i][j]
==0
else
True
printf("| %10.0f",matrix[i][j])
printf("| %10.0f",f)
End For
End For
End For

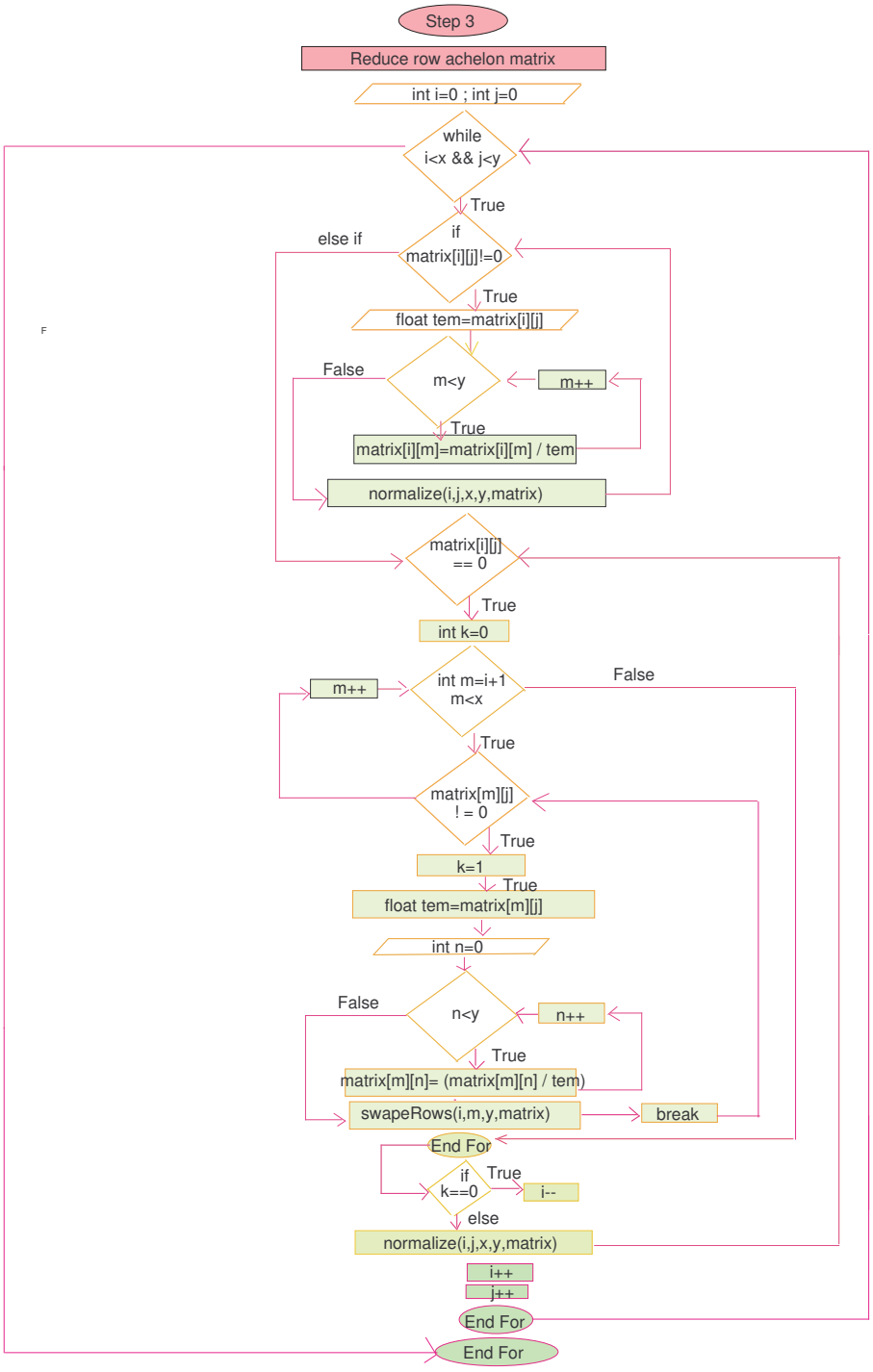

Step 4
matrix Formation
Taking str*equation ,int x
String eq[x-1]
int i = 0
False
i<x-1
i++
True
strcpy(eq[i].field,equation[i].field)
End For
cols=(String*) malloc(x* 400* sizeof(String))
int j = 0
False
i<x-1
i++
True
char *tok=strtok(eq[i].field, "+-")
True
while
tok!=Null
True
if
compareTo
(tok)
True
strcpy(cols[col_count].field, tok)
col_count++
else
tok=strtok(NULL,"+-")
End For
float matrix[x-1][col_count]
int i=0 ; int j=0
False
i< x-1
i++
True
False
j<col
-count
j++
True
matrix[i][j]=0
End For
End For
printf("\n order of final matrix is : %d * %d\n",x-1,col_count)


int i=0 ; int j=0
j< col_count
j++
char *str2=getsimilarof(cols[j].field)
False
i<x-1
i++
True
char* ptr
if ptr=strstr(equation[i].field,str2)) != NULL) || (ptr=strstr(equation[i].field, cols[j].field))!=NULL
False
True
if (ptr== &(equation[i] .field[j]
False
True
matrix[i][j]=1
continue
matrix[i][j]=1
*(ptr -1)=='-'
else if
else
matrix[i][j]=1
continue
End For
End For
printf("\nFinally the Matrix is:-\n\n")
display_2D_array(x-1, col_count, matrix)

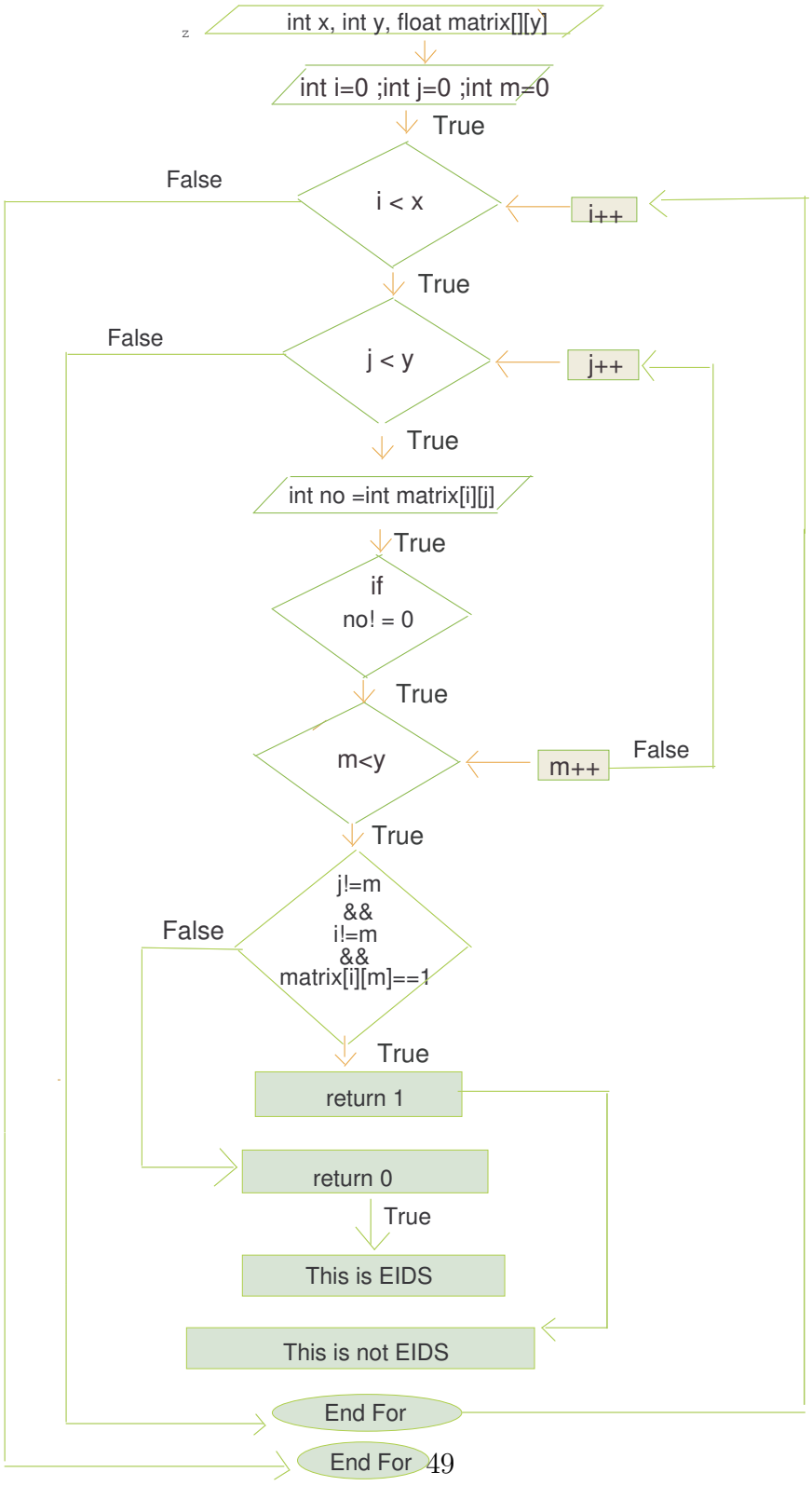
**Step 7:** To Get Reduce Echelon Form




Step 2
swapeRows Function
int x1,int x2,int y, float matrix[][y]
int j=0
False
j<y
j++
True
float temp=matrix[x1][j]
matrix[x1][j]=matrix[x2][j]
matrix[x2][j]=temp
End For


Step 3
Reduce row achelon matrix
int i=0 ; int j=0
while
i<x && j<y
True
if
matrix[i][j]!=0
else if
True
float tem=matrix[i][j]
m<y
False
m++
True
matrix[i][m]=matrix[i][m] / tem
normalize(i,j,x,y,matrix)
matrix[i][j]
== 0
True
int k=0
int m=i+1
m<x
m++
False
True
matrix[m][j]
! = 0
True
k=1
True
float tem=matrix[m][j]
int n=0
n<y
False
n++
True
matrix[m][n]= (matrix[m][n] / tem)
swapeRows(i,m,y,matrix)
break
End For
if
k==0
True
i--
else
normalize(i,j,x,y,matrix)
i++
j++
End For
End For

**Step 8:** To Check EIDS



## 3.2 Examples

In the following examples, we give normal forms of linear EIDS in $M_{33}$ having isolated singularities for $k = 1, ..., 4$ using singular, see Theorem 2.0.10.

**Example 3.2.1.** *We take generic section of the matrix $\alpha_5 : \mathbb{C}^5 \to M_{3,3}$ with the hyperplane $y = w - x$, we get the matrix $\alpha_4 : \mathbb{C}^4 \to M_{3,3}$, given by*

$$\begin{pmatrix} u_1 & u_3 - u_4 & u_3 \\ u_4 & u_5 & u_1 \\ u_3 + u_4 & 2u_1 & -u_5 \end{pmatrix}.$$

*Using Singular we can see that the hypersurface $Y_4 = \alpha_4^{-1}(M^3_{3,3})$ has isolated singularities.*

$> ring\, R = 0, (v, w, x, z), dp;$

$> matrix\, \alpha[3][3] = -v, 2z, w + x, z, v, w, x, w - x, z;$

$> ideal\, f = minor(\alpha, 3);$

$> ideal\, i = jacob(f);$

$> ideal\, k = ideal(f + i);$

$> LIB"solve.lib";$

$> def\, s = solve(k);$

*Consequently, the variety $Y_4$ is an EIDS.*

We take generic section of the matrix $\alpha_4 : \mathbb{C}^4 \to M_{3,3}$ with the hyperplane $v = x$, we get the matrix $\alpha_3 : \mathbb{C}^3 \to M_{3,3}$, given by

$$\begin{pmatrix} u_1 & u_3 - u_1 & u_3 \\ u_1 & u_5 & u_1 \\ u_3 + u_1 & 2u_1 & -u_5 \end{pmatrix}.$$

Using Singular we can see that the hypersurface $Y_3 = \alpha_3^{-1}(M^3_{3,3})$ has isolated singu-

larities.

We take generic section of the matrix $\alpha_3 : \mathbb{C}^3 \to M_{3,3}$ with the hyperplane $w = z$, we get the matrix $\alpha_2 : \mathbb{C}^2 \to M_{3,3}$, given by

$$\begin{pmatrix} u_1 & 0 & u_1 \\ u_1 & u_5 & u_1 \\ 2u_1 & 2u_1 & -u_5 \end{pmatrix}.$$

Using Singular we can see that the hypersurface $Y_2 = \alpha_2^{-1}(M_{3,3}^3)$ has isolated singularities.

We take generic section of the matrix $\alpha_2 : \mathbb{C}^2 \to M_{3,3}$ with the hyperplane $x = z$, we get the matrix $\alpha_1 : \mathbb{C}^1 \to M_{3,3}$, given by

$$\begin{pmatrix} u_1 & 0 & u_1 \\ u_1 & u_1 & u_1 \\ 2u_1 & 2u_1 & -u_1 \end{pmatrix}.$$

Using Singular we can see that the hypersurface $Y_1 = \alpha_1^{-1}(M_{3,3}^3)$ has isolated singularities.

# Bibliography


[1] Ahmed, I. and Ruas, M.A.S. Determinancy of Determinantal Varieties, *Manuscripta Math*, 159 (2019), no. 1-2, 269-278.

[2] Arbarello, E., Cornalba, M., Griffiths, P.A. and Harris, J., Geometry of Algebraic Curves, Vol. I. Grundlehren der Mathematischen Wissenschaften [Fundamental Principles of Mathematical Sciences], vol. 267, Springer, New York, 1985.

[3] Bruce, J.W., Families of Symmetric Matrices, *Mosc. Math. J*, 3 (2003), no. 2, 335-360.

[4] Bruns, W. and Vetter, U., Determinantal Rings, Springer, New York, 1998.

[5] Dimca, A., Topics on Real and Complex Singularities, Vieweg, 1987.

[6] Ebeling, W. and Gusein-Zade, S.M., On Indices of 1-Forms on Determinantal Singularities, *Singularities.Singul. Apple*, 267 (2009), 119-131.

[7] Frühbis-Krüger and Anne, Classification of Simple Space Curve Singularities, *Commun. Algebra*, 27 (1999), no. 8, 3993-4013.

[8] Pereira, M.S., Properties of G-equivalnce of Matrices. arxiv:1711.02156.

[9] Pereira, M.S., Variedades Determinantais e Singularidades de Matrizes, Tese de Doutorado, *ICMC-USP*, 2010.