EE204 Scientific Programming for EE
C Review with Example Codes

Mehmet Çalı

Question

Background

Minors of a matrix is can be calculated from the determinant of sub-matrices which acquired by removing selected row/s and column/s of the original matrix.

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}$$

To calculate minor M_{21} of matrix A, 2^{nd} row and 1^{st} column should be removed

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \qquad M_{21} = \det\left(\begin{bmatrix} a_{12} & a_{13} \\ a_{32} & a_{33} \end{bmatrix}\right)$$

One can extend this example for any row i and column j of matrix A to compute M_{ij}

Question

- Write a function to calculate specified minor of a given matrix

Note: In the laboratory experiments you are generally asked to implement your code with a function prototype restriction

To be able to see different structures of C, this problem is defined in multiple versions which basically correspond to multiple different prototype definitions

Version-1: 2D Automatic Allocation

Prototype constraint:

```
double computeMinor(int r_m, int c_m, int r, double Mat[N][N]);
```

where,

- r_m is row index of the minor
- c_m is column index of the minor
- r is the row and column number of input matrix
- Mat[N][N] is the input square matrix where N is a sufficiently large number defined at the begining of the program

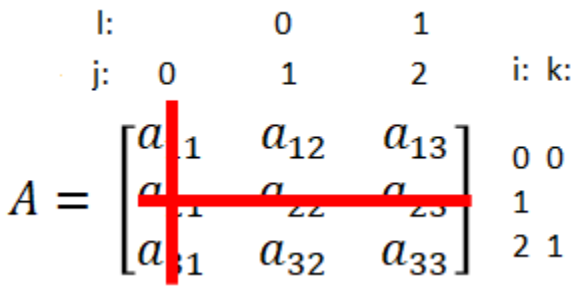
As you can see the function has a return type of double which the output minor result should be assigned to

Assume the determinant function is given with prototype:

```
double determinant(int r, double Mat[N][N]);
```

Function implementation

```
double computeMinor(int r_m, int c_m, int r, double Mat[N][N]){
    int k = 0, l = 0;
    double subMat[N][N];
    for(int i = 0; i < r; i++){
        l = 0;
        for(int j = 0; j < r; j++){
            if((i != (r_m - 1)) && (j != (c_m - 1))){
                subMat[k][l] = Mat[i][j];
                l++;
            }
        }
        if(i != (r_m - 1) ){
            k++;
        }
    }
    printf("sub-mat:\n");
    printMat(r-1, r-1, subMat);
    return determinant(r-1, subMat);
}
```



Main function implementation

```
int main()
{
    double r, det, minor;
    double Mat[N][N];
    printf("Enter the size n of a nxn matrix:\n");
    scanf("%lf",&r);
    getMat(r, r, Mat);

    printMat(r, r, Mat);
    minor = computeMinor(2, 1, r, Mat);
    printf("%f\n", minor);
    return 0;
}
```

Output:

```
Matrix is:
1.000000        2.000000        3.000000
4.000000        5.000000        6.000000
7.000000        8.000000        9.000000
sub-mat:
Matrix is:
2.000000        3.000000
8.000000        9.000000
-6.000000
```

Version-2: Pass by Reference

- Assume the question had a different constraint. Instead of returning the minor directly we output it using pass by reference. So intead of

```
double computeMinor(int r_m, int c_m, int r, double Mat[N][N]);
```

we have the prototype constraint of

```
void computeMinor(int r_m, int c_m, int r, double Mat[N][N], double *minor);
```

Function implementation

Lets modify from main of version 1:

```
double computeMinor(int r_m, int c_m, int r, double Mat[N][N]){
    int k = 0, l = 0;
    double subMat[N][N];
    for(int i = 0; i < r; i++){
        l = 0;
        for(int j = 0; j < r; j++){
            if((i != (r_m - 1)) && (j != (c_m - 1))){
                subMat[k][l] = Mat[i][j];
                l++;
            }
        }
        if(i != (r_m - 1) ){
            k++;
        }
    }
    printf("sub-mat:\n");
    printMat(r-1, r-1, subMat);
    return determinant(r-1, subMat);
}
```

Function implementation

Lets modify from main of version 1:

```
void computeMinor(int r_m, int c_m, int r, double Mat[N][N], double *minor){
    int k = 0, l = 0;
    double subMat[N][N];
    for(int i = 0; i < r; i++){
        l = 0;
        for(int j = 0; j < r; j++){
            if((i != (r_m - 1)) && (j != (c_m - 1))){
                subMat[k][l] = Mat[i][j];
                l++;
            }
        }
        if(i != (r_m - 1) ){
            k++;
        }
    }
    printf("sub-mat:\n");
    printMat(r-1, r-1, subMat);
    return determinant(r-1, subMat);
}
```

Function implementation

Lets modify from main of version 1:

```
void computeMinor(int r_m, int c_m, int r, double Mat[N][N], double *minor){
    int k = 0, l = 0;
    double subMat[N][N];
    for(int i = 0; i < r; i++){
        l = 0;
        for(int j = 0; j < r; j++){
            if((i != (r_m - 1)) && (j != (c_m - 1))){
                subMat[k][l] = Mat[i][j];
                l++;
            }
        }
        if(i != (r_m - 1) ){
            k++;
        }
    }
    printf("sub-mat:\n");
    printMat(r-1, r-1, subMat);
    *minor = determinant(r-1, subMat);
}
```

Main function implementation

Lets modify from main of version 1:

```
int main()
{
    double r, det, minor;
    double Mat[N][N];
    printf("Enter the size n of a nxn matrix:\n");
    scanf("%lf",&r);
    getMat(r, r, Mat);

    printMat(r, r, Mat);
    minor = computeMinor(2, 1, r, Mat);
    printf("%f\n", minor);
    return 0;
}
```

Main function implementation

Lets modify from main of version 1:

```
int main()
{
    double r, det, minor;
    double Mat[N][N];
    printf("Enter the size n of a nxn matrix:\n");
    scanf("%lf",&r);
    getMat(r, r, Mat);

    printMat(r, r, Mat);
    computeMinor(2, 1, r, Mat, &minor);
    printf("%f\n", minor);
    return 0;
}
```

Main function implementation

Lets modify from main of version 1:

```
int main()
{
    double r, det, minor;
    double Mat[N][N];
    printf("Enter the size n of a nxn matrix:\n");
    scanf("%lf",&r);
    getMat(r, r, Mat);

    printMat(r, r, Mat);
    computeMinor(2, 1, r, Mat, &minor);
    printf("%f\n", minor);
    return 0;
}
```

Output:

```
Matrix is:
1.000000        2.000000        3.000000
4.000000        5.000000        6.000000
7.000000        8.000000        9.000000
sub-mat:
Matrix is:
2.000000        3.000000
8.000000        9.000000
-6.000000
```

Version-3: 1D Representation

- Now we alter the prototype constraint such that we can use one dimentional matrix representation. New prototype constraint is:

```
double computeMinor(int r_m, int c_m, int r, double *Mat);
```

or

```
double computeMinor(int r_m, int c_m, int r, double Mat[N]);
```

which are basically the same for this problem as both of them will be used to represent an array

Function implementation

Version-1

```
double computeMinor(int r_m,int c_m,int r,double Mat[][N]){
    int k = 0, l = 0;
    double subMat[N][N];
    for(int i = 0; i < r; i++){
        l = 0;
        for(int j = 0; j < r; j++){
            if((i != (r_m - 1)) && (j != (c_m - 1))){
                subMat[k][l] = Mat[i][j];
                l++;
            }
        }
        if(i != (r_m - 1) ){
            k++;
        }
    }
    printf("sub-mat:\n");
    printMat(r-1, r-1, subMat);
    return determinant(r-1, subMat);
}
```

Version-3

```
double computeMinor(int r_m,int c_m,int r,double *Mat){
    #define subMat(i,j) *(subMat+i*(r-1)+j)
    int k = 0, l = 0;
    double subMat[N];
    for(int i = 0; i < r; i++){
        l = 0;
        for(int j = 0; j < r; j++){
            if((i != (r_m - 1)) && (j != (c_m - 1))){
                subMat(k,l) = Mat(i,j);
                l++;
            }
        }
        if(i != (r_m - 1) ){
            k++;
        }
    }
    printf("sub-mat:\n");
    printMat(r-1, r-1, subMat);
    return determinant(r-1, subMat);
}
```

Define Statement

```
#define subMat(i,j) *(subMat+i*(r-1)+j)
```

(r-1) is the maximum column number of subMat matrix for this example, so replace (r-1) with whatever the maximum column number is in other examples ***(subMat+i*(r-1)+j)** is equivalent to ***(subMat+k)** where **k=i*(r-1)+ j** which is also equivalent to **subMat[k]**

- Memory:

$$subMat[N] = \begin{matrix} subMat[0] \\ subMat[1] \\ subMat[2] \\ subMat[3] \\ subMat[4] \\ subMat[5] \\ subMat[6] \\ subMat[7] \\ subMat[8] \end{matrix}$$

- Virtual Matrix Representation:

$$subMat[N] = \begin{matrix} subMat[0] & subMat[1] & subMat[2] \\ subMat[3] & subMat[4] & subMat[5] \\ subMat[6] & subMat[7] & subMat[8] \end{matrix}$$

For example if we want to access element (i,j) of a mxn matrix we multiply first index with maximum column number and add with second index k=i*n+j
In this example: 4=1*3+1

Main function implementation

```
#include <stdio.h>
#include <stdlib.h>
#define N 100
#define Mat(i,j) *(Mat+i*r+j)


int main()
{
    double r, det, minor;
    double Mat[N];
    printf("Enter the size n of a nxn matrix:\n");
    scanf("%lf",&r);
    getMat(r, r, Mat);

    printMat(r, r, Mat);
    minor = computeMinor(2, 1, r, Mat);
    printf("%f\n", minor);
    return 0;
}
```

Output:

```
Matrix is:
1.000000        2.000000        3.000000
4.000000        5.000000        6.000000
7.000000        8.000000        9.000000
sub-mat:
Matrix is:
2.000000        3.000000
8.000000        9.000000
-6.000000
```

Version-4: Dynamic Memory Allocation

- This time lets add dynamic memory allocation condition to the constraints of one dimensional representation version (version-3 prototype stays the same)

```
double computeMinor(int r_m, int c_m, int r, double *Mat){
    #define subMat(i,j) *(subMat+i*(r-1)+j)
    int k = 0, l = 0;
    double subMat[N];
    for(int i = 0; i < r; i++){
        l = 0;
        for(int j = 0; j < r; j++){
            if((i != (r_m - 1)) && (j != (c_m - 1))){
                subMat(k,l) = Mat(i,j);
                l++;
            }
        }
        if(i != (r_m - 1) ){
            k++;
        }
    }
    printf("sub-mat:\n");
    printMat(r-1, r-1, subMat);
    return determinant(r-1, subMat);
}
```

Version-4: Dynamic Memory Allocation

- This time lets add dynamic memory allocation condition to the constraints of one dimensional representation version (version-3 prototype stays the same)

```
double computeMinor(int r_m, int c_m, int r, double *Mat){
    #define subMat(i,j) *(subMat+i*(r-1)+j)
    int k = 0, l = 0;
    double *subMat = malloc(sizeof(double)*(r-1)*(r-1));
    for(int i = 0; i < r; i++){
        l = 0;
        for(int j = 0; j < r; j++){
            if((i != (r_m - 1)) && (j != (c_m - 1))){
                subMat(k,l) = Mat(i,j);
                l++;
            }
        }
        if(i != (r_m - 1) ){
            k++;
        }
    }
    printf("sub-mat:\n");
    printMat(r-1, r-1, subMat);
    return determinant(r-1, subMat);
}
```

Version-4: Dynamic Memory Allocation

- This time lets add dynamic memory allocation condition to the constraints of one dimensional representation version (version-3 prototype stays the same)

```
double computeMinor(int r_m, int c_m, int r, double *Mat){
    #define subMat(i,j) *(subMat+i*(r-1)+j)
    int k = 0, l = 0, minor;
    double *subMat = malloc(sizeof(double)*(r-1)*(r-1));
    for(int i = 0; i < r; i++){
        l = 0;
        for(int j = 0; j < r; j++){
            if((i != (r_m - 1)) && (j != (c_m - 1))){
                subMat(k,l) = Mat(i,j);
                l++;
            }
        }
        if(i != (r_m - 1) ){
            k++;
        }
    }
    printf("sub-mat:\n");
    printMat(r-1, r-1, subMat);
    return determinant(r-1, subMat);
}
```

Version-4: Dynamic Memory Allocation

- This time lets add dynamic memory allocation condition to the constraints of one dimensional representation version (version-3 prototype stays the same)

```
double computeMinor(int r_m, int c_m, int r, double *Mat){
    #define subMat(i,j) *(subMat+i*(r-1)+j)
    int k = 0, l = 0, minor;
    double *subMat = malloc(sizeof(double)*(r-1)*(r-1));
    for(int i = 0; i < r; i++){
        l = 0;
        for(int j = 0; j < r; j++){
            if((i != (r_m - 1)) && (j != (c_m - 1))){
                subMat(k,l) = Mat(i,j);
                l++;
            }
        }
        if(i != (r_m - 1) ){
            k++;
        }
    }
    printf("sub-mat:\n");
    printMat(r-1, r-1, subMat); minor = determinant(r-1, subMat); free(subMat);
    return determinant(r-1, subMat);
}
```

Version-4: Dynamic Memory Allocation

- This time lets add dynamic memory allocation condition to the constraints of one dimensional representation version (version-3 prototype stays the same)

```
double computeMinor(int r_m, int c_m, int r, double *Mat){
    #define subMat(i,j) *(subMat+i*(r-1)+j)
    int k = 0, l = 0, minor;
    double *subMat = malloc(sizeof(double)*(r-1)*(r-1));
    for(int i = 0; i < r; i++){
        l = 0;
        for(int j = 0; j < r; j++){
            if((i != (r_m - 1)) && (j != (c_m - 1))){
                subMat(k,l) = Mat(i,j);
                l++;
            }
        }
        if(i != (r_m - 1) ){
            k++;
        }
    }
    printf("sub-mat:\n");
    printMat(r-1, r-1, subMat); minor = determinant(r-1, subMat); free(subMat);
    return minor;
}
```

Main function implementation

```
int main()
{
    double r, det, minor;
    double *Mat;
    printf("Enter the size n of a nxn matrix:\n");
    scanf("%lf",&r);
    Mat = malloc(sizeof(double)*r*r);
    getMat(r, r, Mat);

    printMat(r, r, Mat);
    minor = computeMinor(2, 1, r, Mat);
    printf("%f\n", minor);
    return 0;
}
```

Output:

```
Matrix is:
1.000000        2.000000        3.000000
4.000000        5.000000        6.000000
7.000000        8.000000        9.000000
sub-mat:
Matrix is:
2.000000        3.000000
8.000000        9.000000
-6.000000
```

Question-2

- Lets slightly modify the problem by asking sub-matrix that is obtained by removing row r_m and column c_m

Lets start with 2D automatic allocation:

```
double** computeSubMat(int r_m, int c_m, int r, double Mat[N][N]){
    int k = 0, l = 0;
    double subMat[N][N];
    for(int i = 0; i < r; i++){
        l = 0;
        for(int j = 0; j < r; j++){
            if((i != (r_m - 1)) && (j != (c_m - 1))){
                subMat[k][l] = Mat[i][j];
                l++;
            }
        }
        if(i != (r_m - 1) ){
            k++;
        }
    }
    return subMat;
}
```

Local variable cannot be reached outside the scope

What is wrong with this code?

Version-1: 2D Automatic allocation

Lets modify from the wrong approach:

```
double** computeSubMat(int r_m, int c_m, int r, double Mat[N][N]){
    int k = 0, l = 0;
    double subMat[N][N];
    for(int i = 0; i < r; i++){
        l = 0;
        for(int j = 0; j < r; j++){
            if((i != (r_m - 1)) && (j != (c_m - 1))){
                subMat[k][l] = Mat[i][j];
                l++;
            }
        }
        if(i != (r_m - 1) ){
            k++;
        }
    }
    return subMat;
}
```

Version-1: 2D Automatic allocation

Lets modify from the wrong approach:

```
double** computeSubMat(int r_m, int c_m, int r, double Mat[N][N]){
    int k = 0, l = 0;

    for(int i = 0; i < r; i++){
        l = 0;
        for(int j = 0; j < r; j++){
            if((i != (r_m - 1)) && (j != (c_m - 1))){
                subMat[k][l] = Mat[i][j];
                l++;
            }
        }
        if(i != (r_m - 1) ){
            k++;
        }
    }
    return subMat;
}
```

Version-1: 2D Automatic allocation

Lets modify from the wrong approach:

```
void computeSubMat(int r_m, int c_m, int r, double Mat[N][N], double subMat[N][N]){
    int k = 0, l = 0;

    for(int i = 0; i < r; i++){
        l = 0;
        for(int j = 0; j < r; j++){
            if((i != (r_m - 1)) && (j != (c_m - 1))){
                subMat[k][l] = Mat[i][j];
                l++;
            }
        }
        if(i != (r_m - 1) ){
            k++;
        }
    }
    return subMat;
}
```

Version-1: 2D Automatic allocation

Lets modify from the wrong approach:

```
void computeSubMat(int r_m, int c_m, int r, double Mat[N][N], double subMat[N][N]){
    int k = 0, l = 0;

    for(int i = 0; i < r; i++){
        l = 0;
        for(int j = 0; j < r; j++){
            if((i != (r_m - 1)) && (j != (c_m - 1))){
                subMat[k][l] = Mat[i][j];
                l++;
            }
        }
        if(i != (r_m - 1) ){
            k++;
        }
    }

}
```

Main Function Implementation

- Lets modify from main of first question

```
int main()
{
    double r, det, minor;
    double Mat[N][N];
    printf("Enter the size n of a nxn matrix:\n");
    scanf("%lf",&r);
    getMat(r, r, Mat);

    printMat(r, r, Mat);
    minor = computeMinor(2, 1, r, Mat);
    printf("%f\n", minor);
    return 0;
}
```

Main Function Implementation

- Lets modify from main of first question

```
int main()
{   double sMat[N][N];
    double r, det, minor;
    double Mat[N][N];
    printf("Enter the size n of a nxn matrix:\n");
    scanf("%lf",&r);
    getMat(r, r, Mat);

    printMat(r, r, Mat);
    minor = computeMinor(2, 1, r, Mat);
    printf("%f\n", minor);
    return 0;
}
```

Main Function Implementation

- Lets modify from main of first question

```
int main()
{   double sMat[N][N];
    double r, det, minor;
    double Mat[N][N];
    printf("Enter the size n of a nxn matrix:\n");
    scanf("%lf",&r);
    getMat(r, r, Mat);

    printMat(r, r, Mat);
    computeSubMat(2, 1, r, Mat, sMat);
    printf("%f\n", minor);
    return 0;
}
```

Main Function Implementation

- Lets modify from main of first question

```
int main()
{   double sMat[N][N];
    double r, det, minor;
    double Mat[N][N];
    printf("Enter the size n of a nxn matrix:\n");
    scanf("%lf",&r);
    getMat(r, r, Mat);

    printMat(r, r, Mat);
    computeSubMat(2, 1, r, Mat, sMat);
    printMat(r-1, r-1, sMat);
    return 0;
}
```

Main Function Implementation

- Lets modify from main of first question

```
int main()
{   double sMat[N][N];
    double r, det, minor;
    double Mat[N][N];
    printf("Enter the size n of a nxn matrix:\n");
    scanf("%lf",&r);
    getMat(r, r, Mat);

    printMat(r, r, Mat);
    computeSubMat(2, 1, r, Mat, sMat);
    printMat(r-1, r-1, sMat);
    return 0;
}
```

```
Matrix is:
1.000000        2.000000        3.000000
4.000000        5.000000        6.000000
7.000000        8.000000        9.000000
Matrix is:
2.000000        3.000000
8.000000        9.000000
```

Version-2: 1D Dynamic allocation

- Lets modify from the question-1 dynamic allocation:

```
double computeMinor(int r_m, int c_m, int r, double *Mat){
    #define subMat(i,j) *(subMat+i*(r-1)+j)
    int k = 0, l = 0;
    double *subMat = malloc(sizeof(double)*(r-1)*(r-1));
    for(int i = 0; i < r; i++){
        l = 0;
        for(int j = 0; j < r; j++){
            if((i != (r_m - 1)) && (j != (c_m - 1))){
                subMat(k,l) = Mat(i,j);
                l++;
            }
        }
        if(i != (r_m - 1) ){
            k++;
        }
    }
    printf("sub-mat:\n");
    printMat(r-1, r-1, subMat);
    return determinant(r-1, subMat);
}
```

Version-2: 1D Dynamic allocation

- Lets modify from the question-1 dynamic allocation:

```
double computeMinor(int r_m, int c_m, int r, double *Mat){
    #define subMat(i,j) *(subMat+i*(r-1)+j)
    int k = 0, l = 0;
    double *subMat = malloc(sizeof(double)*(r-1)*(r-1));
    for(int i = 0; i < r; i++){
        l = 0;
        for(int j = 0; j < r; j++){
            if((i != (r_m - 1)) && (j != (c_m - 1))){
                subMat(k,l) = Mat(i,j);
                l++;
            }
        }
        if(i != (r_m - 1) ){
            k++;
        }
    }


    return determinant(r-1, subMat);
}
```

Version-2: 1D Dynamic allocation

- Lets modify from the question-1 dynamic allocation:

```
double computeMinor(int r_m, int c_m, int r, double *Mat){
    #define subMat(i,j) *(subMat+i*(r-1)+j)
    int k = 0, l = 0;
    double *subMat = malloc(sizeof(double)*(r-1)*(r-1));
    for(int i = 0; i < r; i++){
        l = 0;
        for(int j = 0; j < r; j++){
            if((i != (r_m - 1)) && (j != (c_m - 1))){
                subMat(k,l) = Mat(i,j);
                l++;
            }
        }
        if(i != (r_m - 1) ){
            k++;
        }
    }


    return subMat;
}
```

Version-2: 1D Dynamic allocation

- Lets modify from the question-1 dynamic allocation:

```
double* computeSubMat(int r_m, int c_m, int r, double *Mat){
    #define subMat(i,j) *(subMat+i*(r-1)+j)
    int k = 0, l = 0;
    double *subMat = malloc(sizeof(double)*(r-1)*(r-1));
    for(int i = 0; i < r; i++){
        l = 0;
        for(int j = 0; j < r; j++){
            if((i != (r_m - 1)) && (j != (c_m - 1))){
                subMat(k,l) = Mat(i,j);
                l++;
            }
        }
        if(i != (r_m - 1) ){
            k++;
        }
    }


    return subMat;
}
```

Main Function Implementation

- Lets modify from the question-1 dynamic allocation:

```
int main()
{
    double r, det, minor;
    double *Mat;
    printf("Enter the size n of a nxn matrix:\n");
    scanf("%lf",&r);
    Mat = malloc(sizeof(double)*r*r);
    getMat(r, r, Mat);

    printMat(r, r, Mat);
    minor = computeMinor(2, 1, r, Mat);
    printf("%f\n", minor);
    free(Mat);
    return 0;
}
```

Main Function Implementation

- Lets modify from the question-1 dynamic allocation:

```
int main()
{
    double r, det, *sMat;
    double *Mat;
    printf("Enter the size n of a nxn matrix:\n");
    scanf("%lf",&r);
    Mat = malloc(sizeof(double)*r*r);
    getMat(r, r, Mat);

    printMat(r, r, Mat);
    minor = computeMinor(2, 1, r, Mat);
    printf("%f\n", minor);
    free(Mat);
    return 0;
}
```

Main Function Implementation

- Lets modify from the question-1 dynamic allocation:

```
int main()
{
    double r, det, *sMat;
    double *Mat;
    printf("Enter the size n of a nxn matrix:\n");
    scanf("%lf",&r);
    Mat = malloc(sizeof(double)*r*r);
    getMat(r, r, Mat);

    printMat(r, r, Mat);
    sMat = computeSubMat(2, 1, r, Mat);
    printf("%f\n", minor);
    free(Mat);
    return 0;
}
```

Main Function Implementation

- Lets modify from the question-1 dynamic allocation:

```
int main()
{
    double r, det, *sMat;
    double *Mat;
    printf("Enter the size n of a nxn matrix:\n");
    scanf("%lf",&r);
    Mat = malloc(sizeof(double)*r*r);
    getMat(r, r, Mat);

    printMat(r, r, Mat);
    sMat = computeSubMat(2, 1, r, Mat);
    printMat(r-1, r-1, sMat);
    free(Mat);
    return 0;
}
```

Main Function Implementation

- Lets modify from the question-1 dynamic allocation:

```
int main()
{
    double r, det, *sMat;
    double *Mat;
    printf("Enter the size n of a nxn matrix:\n");
    scanf("%lf",&r);
    Mat = malloc(sizeof(double)*r*r);
    getMat(r, r, Mat);

    printMat(r, r, Mat);
    sMat = computeSubMat(2, 1, r, Mat);
    printMat(r-1, r-1, sMat);
    free(Mat); free(sMat);
    return 0;
}
```

Version-3 Static 2D Representation

- Lets modify from failed attempt at version-1

```
double** computeSubMat(int r_m, int c_m, int r, double Mat[N][N]){
    int k = 0, l = 0;
    double subMat[N][N];
    for(int i = 0; i < r; i++){
        l = 0;
        for(int j = 0; j < r; j++){
            if((i != (r_m - 1)) && (j != (c_m - 1))){
                subMat[k][l] = Mat[i][j];
                l++;
            }
        }
        if(i != (r_m - 1) ){
            k++;
        }
    }
    return subMat;
}
```

Version-3 Static 2D Representation

- Lets modify from failed attempt at version-1

```
double** computeSubMat(int r_m, int c_m, int r, double Mat[N][N]){
    int k = 0, l = 0;
    static double subMat[N][N];
    for(int i = 0; i < r; i++){
        l = 0;
        for(int j = 0; j < r; j++){
            if((i != (r_m - 1)) && (j != (c_m - 1))){
                subMat[k][l] = Mat[i][j];
                l++;
            }
        }
        if(i != (r_m - 1) ){
            k++;
        }
    }
    return subMat;
}
```

Main Function Implementation

- Lets modify from main of version 1

```
int main()
{
    double sMat[N][N];
    double r, det;
    double Mat[N][N];
    printf("Enter the size n of a nxn matrix:\n");
    scanf("%lf",&r);
    getMat(r, r, Mat);

    printMat(r, r, Mat);
    computeSubMat(2, 1, r, Mat, sMat);
    printMat(r-1, r-1, sMat);
    return 0;
}
```

Main Function Implementation

- Lets modify from main of version 1

```
int main()
{
    double sMat[N][N];
    double r, det;
    double Mat[N][N];
    printf("Enter the size n of a nxn matrix:\n");
    scanf("%lf",&r);
    getMat(r, r, Mat);

    printMat(r, r, Mat);
    sMat = computeSubMat(2, 1, r, Mat);
    printMat(r-1, r-1, sMat);
    return 0;
}
```

Main Function Implementation

- Lets modify from main of version 1

```
int main()
{
    double **sMat;
    double r, det;
    double Mat[N][N];
    printf("Enter the size n of a nxn matrix:\n");
    scanf("%lf",&r);
    getMat(r, r, Mat);

    printMat(r, r, Mat);
    sMat = computeSubMat(2, 1, r, Mat);
    printMat(r-1, r-1, sMat);
    return 0;
}
```