**ASSIGNMENT-11**

**1.AIM**

Write a program in C to implement lower and upper triangular matrix using one dimensional array.

**2.ALGORITHM**

**Algorithm** Is\_Upper\_Triangular\_Matrix

**Input:** A pointer ‘arr’ to the matrix and the dimension ‘size’ of the matrix

**Output:** If the input matrix is a upper triangular matrix , nothing is returned else a error message is shown and the program terminates.

**Remarks:**It is assumed that the array is not empty.

**Steps:**

1. **For**(i=2 to size) //Traversing the lower triangular part
2. **For**(j=0 to i)
3. **If**(arr[i][j]≠0) **then** //non zero lower triangular element
4. print“Input matrix is not upper triangular matrix”
5. **Exit** // terminate the program
6. **Return** // If program does not exit, input matrix is upper triangular
7. **Stop**

**Algorithm** Is\_Lower\_Triangular\_Matrix

**Input:** A pointer ‘arr’ to the matrix and the dimension ‘size’ of the matrix

**Output:** If the input matrix is a lower triangular matrix , nothing is returned else a error message is shown and the program terminates.

**Remarks:**It is assumed that the array is not empty.

**Steps:**

1. **For**(i=2 to size) **do** //Traversing the upper triangular part
2. **For**(j=0 to i) **do**
3. **If**(arr[j][i]≠0) **then** //non zero upper triangular element
4. **print**“Input matrix is not lower triangular matrix”
5. **Exit** // terminate the program
6. **Return** // If program does not exit, input matrix is lower triangular
7. **Stop**

**Algorithm** Make\_Upper\_Triangular\_List

**Input:** A pointer ‘arr’ to the matrix , the dimesion ‘size’ and the pointer ‘list’ to the one dimesional array in which the upper triangular part is to be stored.

**Output:** The upper triangular part of ‘arr’ is fed into the one dimesional array ‘list’ by reference

**Remarks:** Size of the one dimesional array = (size(size+1))/2

**Steps:**

1. **For**(i=1 to size) **do** // Traversing the upper triangular part
2. **For**(j=i to size) **do**
3. list[k]=arr[i][j] // Feeding the one dimesional array
4. k = k+1
5. **Return**
6. **Stop**

**Algorithm** Make\_Lower\_Triangular\_List

**Input:** A pointer ‘arr’ to the matrix , the dimesion ‘size’ and the pointer ‘list’ to the one dimesional array in which the lower triangular part is to be stored.

**Output:** The lower triangular part of ‘arr’ is fed into the one dimesional array ‘list’ by reference

**Remarks:** Size of the one dimesional array = (size(size+1))/2

**Steps:**

1. **For**(i=1 to size) **do** // Traversing the lower triangular part
2. **For**(j=i to (i+1) **do**
3. list[k]=arr[i][j] // Feeding the one dimesional array
4. k=k+1
5. **Return**
6. **Stop**

**Algorithm** Reconstruct\_Upper\_Triangular\_Matrix

**Input:** A pointer ‘arr’ to the one dimesional array in which the upper triangular part is stored and the dimesion ‘size’ of the matrix to be reconstructed.

**Output:** The original input matrix is displayed using only the one dimesional array that holds the upper triangular part.

**Remarks:**Matrix is not recreated physically , it is only displayed.

**Steps:**

1. limit=size
2. **For**(i=1 to size) **do** // For non upper diagonal positions
3. **For**(j=limit to size)
4. **Print** “0” // display 0
5. **For**(k=1 to limit) // For upper diagonal positions
6. **Print** arr[l] // Display the contents of 1-D array
7. l=l+1
8. limit=limit-1
9. **Stop**

**Algorithm** Reconstruct\_Lower\_Triangular\_Matrix

**Input:** A pointer ‘arr’ to the one dimesional array in which the lower triangular part is stored and the dimesion ‘size’ of the matrix to be reconstructed.

**Output:** The original input matrix is displayed using only the one dimesional array that holds the upper triangular part.

**Remarks:**Matrix is not recreated physically , it is only displayed.

**Steps:**

1. limit=size
2. **For**(i=1 to size) **do** // For lower triangular positions
3. **For**(j=(limit-1) to size)
4. **Print** arr[l] // Display the contents of 1-D array
5. l=l+1
6. **For**(k=2 to limit) // For non lower triagular positions
7. **Print** “0” // Display 0
8. limit=limit-1
9. **Stop**

**3.SOURCE CODE**

#include<stdio.h>

#include<stdlib.h>

// function for input validation

void validate(int row,int col)

{

if(row!=col)

{

printf("Triangular Matrix Must Be A Square Matrix!\nPlease Try Again\n");

exit(1);

}

return;

}

// function to ask the user whether to continue with new input

void prompt(void)

{

int ch;

while(1)

{

printf("Press 1 to continue OR press 0 to exit: ");

scanf("%d",&ch);

if(ch==1)

return;

else if(ch==0)

exit(1);

printf("\nInvalid Input, Please Try Again\n");

}

}

// function to take input in a matrix

void getmat(int(\*arr)[20],int row,int col)

{

int i,j;

for(i=0;i<row;i++)

for(j=0;j<col;j++)

scanf("%d",&arr[i][j]);

}

//function to display a matrix

void dispmat(int(\*arr)[20],int row,int col)

{

int i,j;

for(i=0;i<row;i++)

{

for(j=0;j<col;j++)

printf("%d\t",arr[i][j]);

printf("\n");

}

}

//function to check whether input matrix is a upper triangular matrix

void isuppertri(int(\*arr)[20],int row,int col)

{

int i,j;

for(i=1;i<row;i++) //traversing the lower triangular part

for(j=0;j<i;j++)

if(arr[i][j]!=0) // if an element is not zero

{

printf("\nEntered Matrix Is Not A Upper Triangular Matrix\n");

exit(1); // terminate the program

}

printf("\nEntered Matrix Is A Upper Triangular Matrix");

}

//function to check whether input matrix is a lower triangular matrix

void islowertri(int(\*arr)[20],int row,int col)

{

int i,j;

for(i=1;i<row;i++) //traversing the upper triangualar part

for(j=0;j<i;j++)

if(arr[j][i]!=0) //if an element is not zero

{

printf("\nEntered Matrix Is Not A Lower Triangular Matrix\n");

exit(1); //terminate the program

}

printf("\nEntered Matrix Is A Lower Triangular Matrix");

}

//function to feed upper triangular matrix in 1-D array

void makeupperlist(int\*list,int(\*arr)[20],int row,int col)

{

int i,j,k=0;

for(i=0;i<row;i++) //traversing the upper triangular part

for(j=i;j<col;j++)

{

list[k]=arr[i][j]; //feeding the 1-D array

k++;

}

}

//function to feed lower triangular matrix in 1-D array

void makelowerlist(int\*list,int(\*arr)[20],int row,int col)

{

int i,j,k=0;

for(i=0;i<row;i++) //traversing the lower triangular part

for(j=0;j<(i+1);j++)

{

list[k]=arr[i][j]; //feeding the 1-D array

k++;

}

}

//function to display a 1-D array

void displist(int\*arr,int size)

{

int i;

for(i=0;i<size;i++)

printf("%d ",arr[i]);

}

//function to reconstruct matrix from 1-D array

void reconsuppmat(int\* arr,int row)

{

int i,j,k,l=0,limit;

limit=row;

for(i=0;i<row;i++)

{

for(j=limit;j<row;j++) //for non upper triangular positions

printf("0\t"); //display 0

for(k=0;k<limit;k++) // for upper triangular positions

{

printf("%d\t",arr[l]); //display contents of 1-D array

l++;

}

limit--;

printf("\n");

}

}

//function to reconstruct matrix from 1-D array

void reconslowmat(int\* arr,int row)

{

int i,j,k,l=0,lim=row;

for(i=0;i<row;i++)

{

for(j=(lim-1);j<row;j++) //for lower triangular positions

{

printf("%d\t",arr[l]); //display contents of 1-D array

l++;

}

for(k=1;k<lim;k++) //for non lower triangular positions

printf("0\t"); // display 0

lim--;

printf("\n");

}

}

int main(void)

{

int ch,row,col,arr[20][20],\*list,size;

while(1)

{

printf("Enter Number Of Rows: ");

scanf("%d",&row);

printf("Enter Number Of Columns: ");

scanf("%d",&col);

validate(row,col);

printf("\nEnter %d elements of the matrix: \n",row\*col);

getmat(arr,row,col);

printf("\nEntered Matrix: \n");

dispmat(arr,row,col);

printf("\nMenu: ");

printf("\n1.Upper Triangular Matrix\n2.Lower Triangular Matrix");

printf("\nEnter Your Choice: ");

scanf("%d",&ch);

switch(ch)

{

case 1:

isuppertri(arr,row,col);

size=(row\*(row+1))/2; //length of 1-D array

list=(int\*)calloc(size,sizeof(int));

makeupperlist(list,arr,row,col);

printf("\nGenerated List: ");

displist(list,size);

printf("\nReconstructed Matrix:\n");

reconsuppmat(list,row);

prompt();

break;

case 2:

islowertri(arr,row,col);

size=(row\*(row+1))/2; //length of 1-D array

list=(int\*)calloc(size,sizeof(int));

makelowerlist(list,arr,row,col);

printf("\nGenerated List: ");

displist(list,size);

printf("\nReconstructed Matrix:\n");

reconslowmat(list,row);

prompt(); //ask the user whether to exit

return 0;

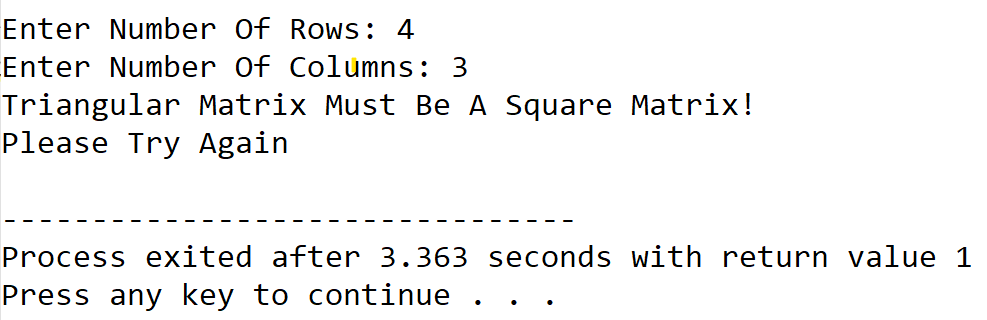
}

}

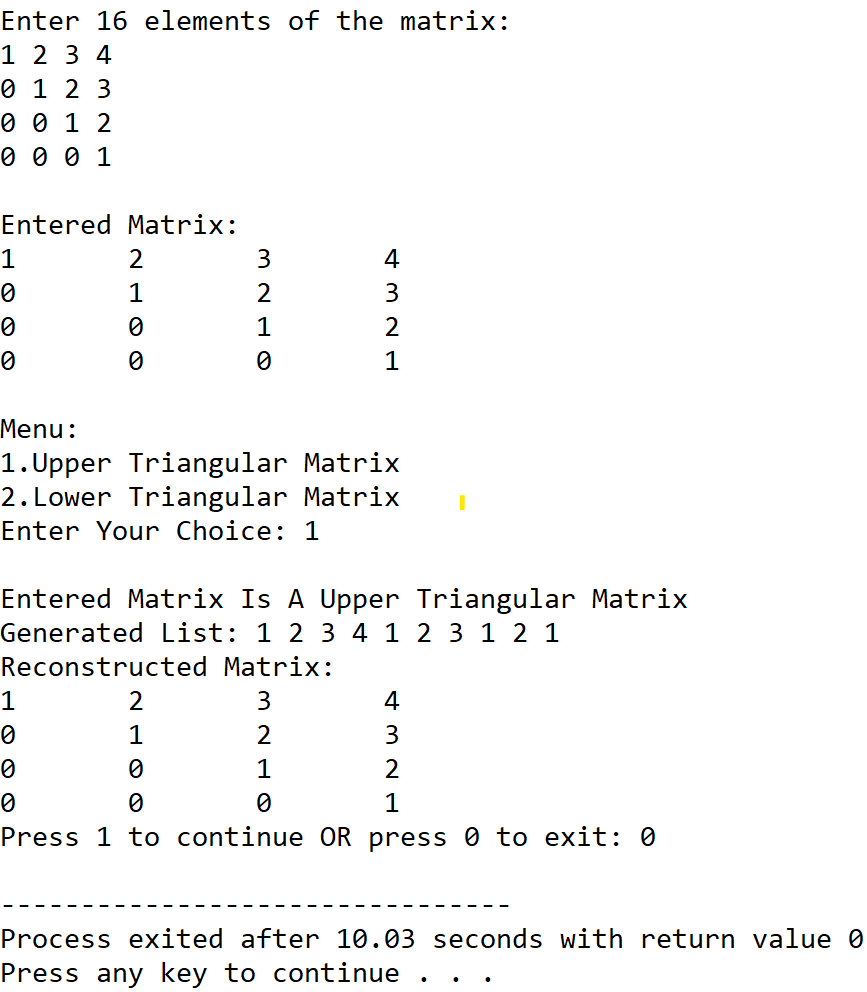
}

**4.OUTPUT**

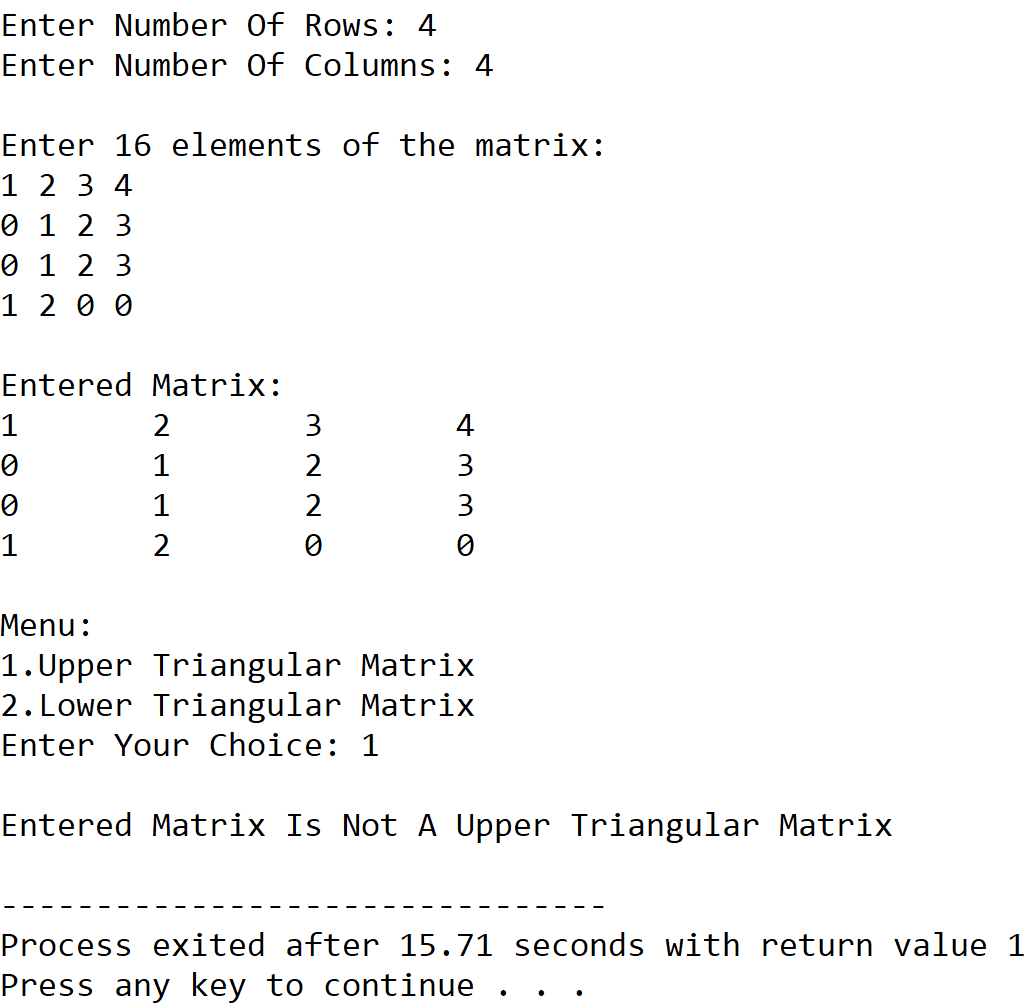
**SET 1:** Input matrix is not a square matrix



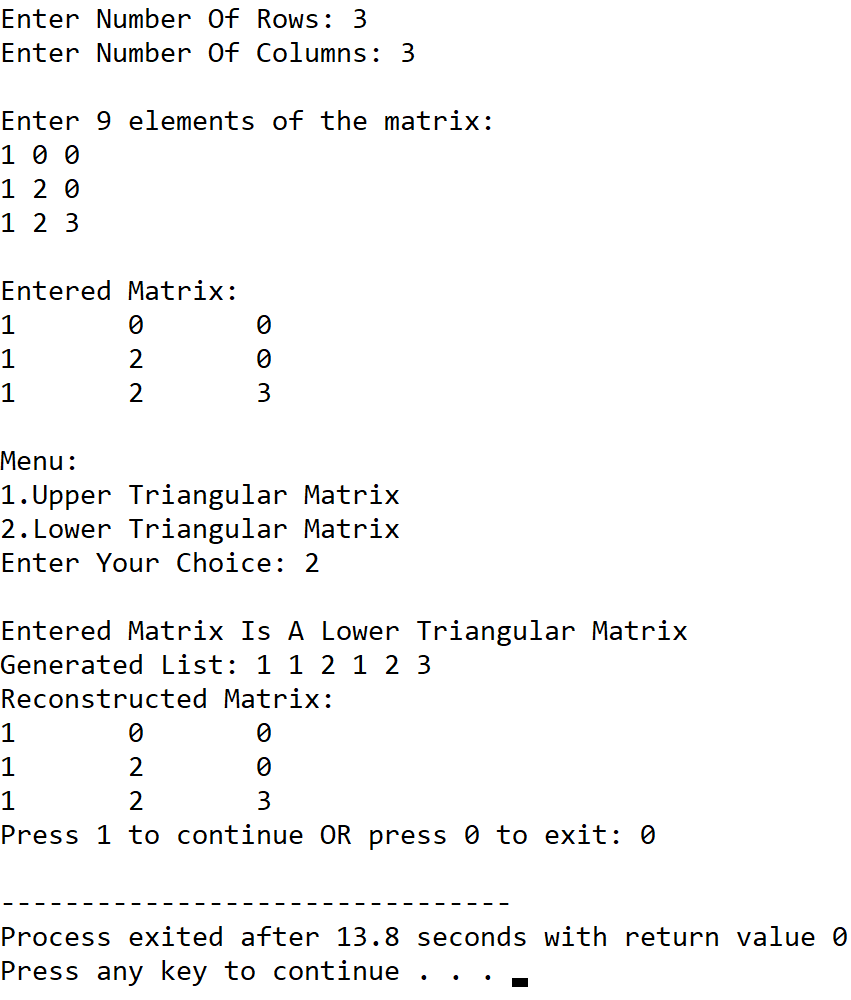
**SET 2:** Input matrix is an upper triangular matrix



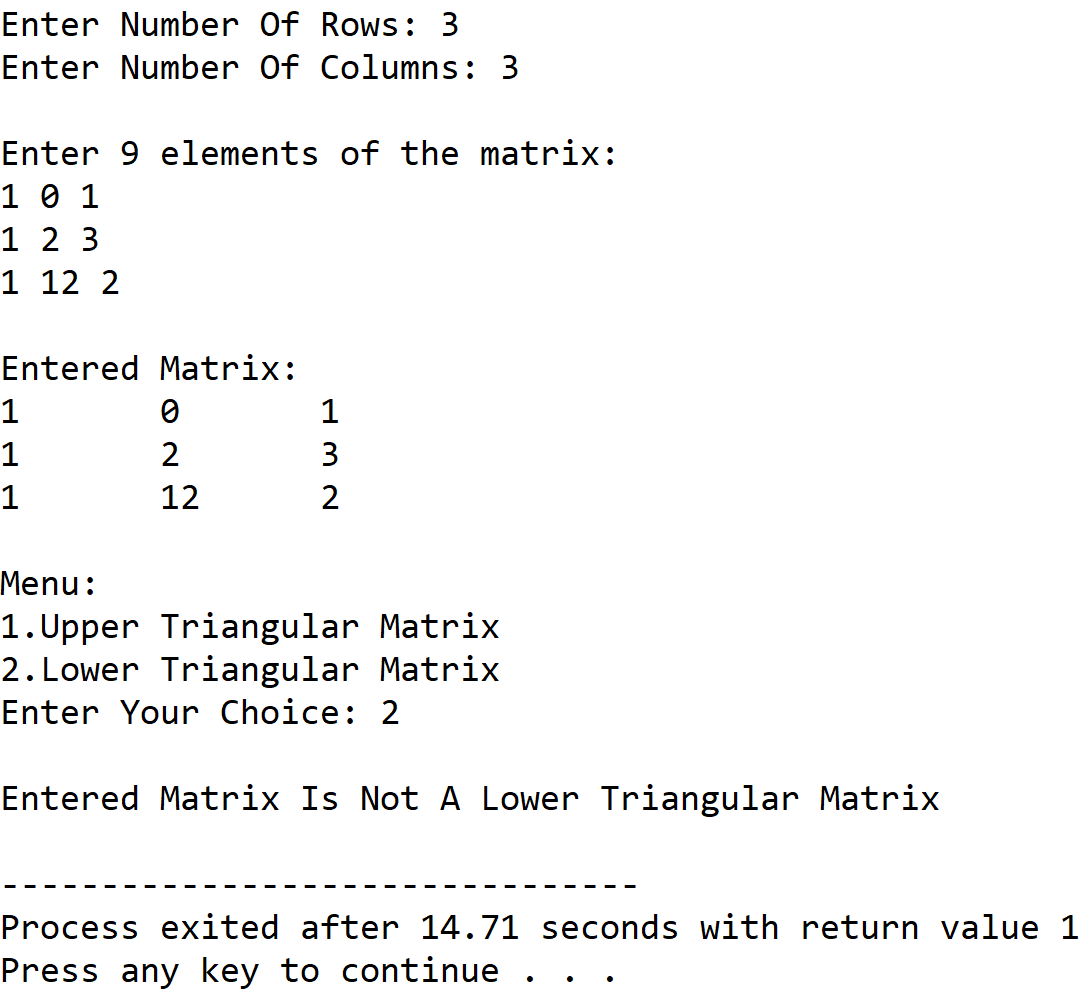
**SET 3:** Input matrix is not an upper triangular matrix



**SET 4:** Input matrix is a lower triangular matrix



**SET 5:** Input matrix is not a lower triangular matrix



**5.DISCUSSIONS**

**A. Variable Description**

* **row,col:** Dimesion of the matrix.
* **arr:** Two dimensional array to hold the matrix.
* **\*list:** pointer to a one dimesional array.
* **size:** length of the one dimesional array.
* **ch:** variable to receive user’s choice in switch-case-default.

**B. Limitations**

* The program uses a one dimensional array to hold the triangular sections of the matrix, as array is a static data structure ,it’s size cannot be altered once it its constructed in the memory.
* The two dimensional array is not dynamically allocated and thus leads to inefficient use of memory.

**C. Uses**

* The program showcases that triangular matrices can be represented by using less scomputer memory without any loss in information if only the upper triangular or lower triangular part is stored in a one dimesional array.This displays how a particular data structure can be represented in tne memory in a much more optimized way as compared to traditional methods.

**D. Future Scope**

* The two dimesional array can be dynamically allocated for more efficient memory use.
* The one dimensional array can be replaced with a linked list for more efficient memory use