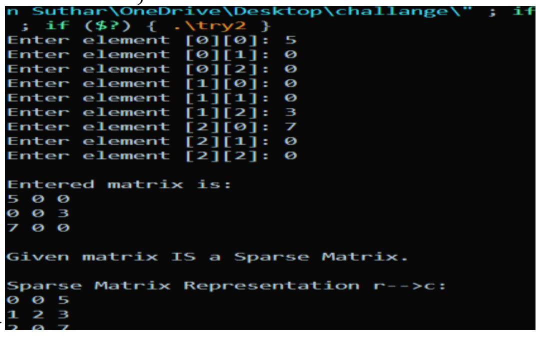
Practical 2

A) Write a Program

i) implement an Array representation of the sparse matrices

Code:

```
#include <stdio.h>
int main()
{
   int arr[3][3];
   int i, j;
   int count = 0; // count of zeros
   int size = 0;  // count of non-zeros

   // Input matrix
   for (i = 0; i < 3; i++)
   {
      for (j = 0; j < 3; j++)
      {
         printf("Enter element [%d][%d]: ", i, j);
         scanf("%d", &arr[i][j]);
      }
   }
   //  print part
   printf("\nEntered matrix is:\n");
   for (i = 0; i < 3; i++)
   {
      for (j = 0; j < 3; j++)
      {
         printf("%d ", arr[i][j]);
      }
      printf("\n");
   }
   // Count zeros and non-zeros
   for (i = 0; i < 3; i++)
   {
      for (j = 0; j < 3; j++)
      {
         if (arr[i][j] == 0)
            count++;
         else
            size++;
      }
   }
   //  if it's a sparse matrix
   if (size > count)
   {
      printf("\nGiven matrix is NOT a Sparse Matrix.\n");
   }
   else
   {
      printf("\nGiven matrix IS a Sparse Matrix.\n");
      int sparse[size][3];
      int t = 0;
      // Fill sparse matrix
      for (i = 0; i < 3; i++)
      {
         for (j = 0; j < 3; j++)
         {
            if (arr[i][j] != 0)
            {
               sparse[t][0] = i;
               sparse[t][1] = j;
               sparse[t][2] = arr[i][j];
               t++;
            }
         }
      }
      // Print sparse matrix
      printf("\nSparse Matrix Representation r-->c:\n");
      for (i = 0; i < size; i++)
      {
         printf("%d %d %d\n", sparse[i][0], sparse[i][1], sparse[i][2]);
      }
   }
   return 0;
}
```



```
n Suthar\OneDrive\Desktop\challange\" ; if
 ; if ($?) { .\try2 }
Enter element [0][0]: 5
Enter element [0][1]: 0
Enter element [0][2]: 0
Enter element [1][0]: 0
Enter element [1][1]: 0
Enter element [1][2]: 3
Enter element [2][0]: 7
Enter element [2][1]: 0
Enter element [2][2]: 0

Entered matrix is:
5 0 0
0 0 3
7 0 0

Given matrix IS a Sparse Matrix.

Sparse Matrix Representation r-->c:
0 0 5
1 2 3
2 0 7
```

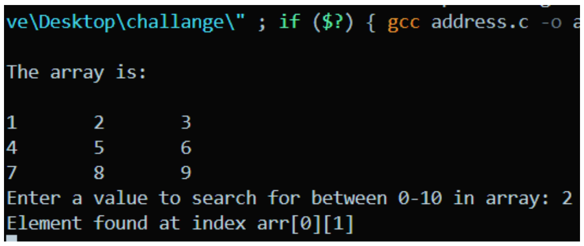
Output:

ii)To search the element in 2-D array.

Code:

```
#include<stdio.h>

#include<conio.h>

int main()
{
int arr[3][3]={{1,2,3},{4,5,6},{7,8,9}};
int i,j,x;
printf("The array is:\n");
for(i=0;i<3;i++)
{
printf("\n");
for(j=0;j<3;j++)
printf("%d\t",arr[i][j]);
}
printf("\nEnter a value to search for between 0-10 in array: ");
scanf("%d",&x);
for(i=0;i<3;i++)
{
for(j=0;j<3;j++)
{
if(arr[i][j]==x){
printf("Element found at index arr[%d][%d]\n",i,j);
break;
}}}
getch();
return 0;
}
```

Output:



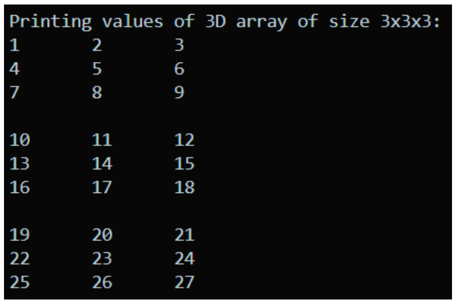
```
ve\Desktop\challange\" ; if ($?) { gcc address.c -o a

The array is:

1       2       3
4       5       6
7       8       9
Enter a value to search for between 0-10 in array: 2
Element found at index arr[0][1]
```

iii) To create, initialize and print the values of the 3-D array.

Code:

```
#include <stdio.h>
#include <stdlib.h> // for system()

int main() {
    int arr[3][3][3];
    int i, j, k, value = 1;

    /
    system("cls");
    printf("Printing values of 3D array of size 3x3x3:\n");
    // Assign values
    for (i = 0; i < 3; i++) {
        for (j = 0; j < 3; j++) {
            for (k = 0; k < 3; k++) {
                arr[i][j][k] = value++;
            }
        }
    }

    // Print values
    for (i = 0; i < 3; i++) {
        for (j = 0; j < 3; j++) {
            for (k = 0; k < 3; k++) {
                printf("%d\t", arr[i][j][k]);
            }
            printf("\n");
        }
        printf("\n");
    }

    return 0;
}
```

Output:



```
Printing values of 3D array of size 3x3x3:
1       2       3
4       5       6
7       8       9

10      11      12
13      14      15
16      17      18

19      20      21
22      23      24
25      26      27
```

B) Discuss following concepts with an example

i) Address Calculation of 1-D Array.

In memory, arrays are stored in contiguous memory locations. The address of an element in 1D arrays can be calculated by using the formula:

Address of A[i]=Base Address+(i × Size of Each Element)

Where,
Base address = starting address of array in memory
i = index of element
size =size of each array element in bytes

Example
int arr[5]= {12,45,56,78,90}
Assume : - base address = 1000
- size of int = 2
Find address of arr[3].

Therefore,
Address = 1000 + (3 x 2) = 1006
Address of element at arr[3] is 1006.

ii) Address Calculation of 2-D Array using Row Major and Column Major Order.

In C (and most languages), a 2-D array is stored in contiguous memory. The way elements are laid out depends on whether the storage is row-major or column-major.

Take: int arr[3][4]; \ Base address=1000

Calculating address of an element in 2D using row major

Row major ordering assigns successive elements, moving across the rows and then down the next row, to successive memory locations. In simple language, the elements of an array are stored in a Row-Wise fashion.
To calculate address of an element using row major, we use the formula:

Address of A[i][j] = B + W * ((j – LC) * N + (i – LR))

Where,
i = Row Subset of an element whose address to be found,
j = Column Subset of an element whose address to be found,
B = Base address,
W = Storage size of one element store in any array(in byte),
LR = Lower Limit of row/start row index of matrix(If not given assume it as zero),
LC = Lower Limit of column/start column index of matrix(If not given assume it as zero)
N = Number of columns given in the matrix.

Example:
Find address of element at index arr[2][3].
In this,
i= 2
j= 3
B=1000
W= 2 bytes
LR= 0
LC= 0
N= 4
Therefore,
Address of A[3][4] = 1000 + 2 * ((3-0) * 4 + (2-0))
= 1000+2(14)
=1028
The address of element at arr[2][3] is 1028.

Calculating address of an element in 2D using column major

Column major ordering assigns successive elements, moving across the columns and then down the next column, to successive memory locations. In simple language, the elements of an array are stored in a column-Wise fashion.
To calculate address of an element using row major, we use the formula:

Address of A[i][j] = B + W * ((j – LC) * M + (i – LR))

Where,
i = Row Subset of an element whose address to be found,
j = Column Subset of an element whose address to be found,

B = Base address,
W = Storage size of one element store in any array(in byte),
LR = Lower Limit of row/start row index of matrix(If not given assume it as zero),
LC = Lower Limit of column/start column index of matrix(If not given assume it as zero)
M = Number of rows given in the matrix.

<u>Example</u>

Find address of element at index arr[2][3] using column major.
In this,
i= 2
j= 3
B=1000
W= 2 bytes
LR= 0
LC= 0
M= 4Therefore,
Address of A[3][4] = 1000 + 2 * ((3-0) * 3+ (2-0))
= 1000+2(11)
=1022
The address of element at arr[2][3] is 1022.