CHAPTER-9

ARRAYS

VERY SHORT/SHORT ANSWER QUESTIONS

1. **Define the term data structure and state its significance.**

Ans. A data structure is a named group of data of different data types which can be processed as a single unit. Data structures are very important in a computer system, as these not only allow the user to combine various data types in a group but also allow processing of the group as a single unit thereby making things much simpler and easier.

2. **Differentiate between one-dimensional and two-dimensional arrays.**

Ans.

One-dimensional array	Two-dimensional array
A one-dimensional array is a group of elements having same data type and same name.	A two-dimensional array is an array in which each element is itself a 1-D array.
There is no rows and columns in one-dimensional array.	There is a concept of rows and columns in two-dimensional array.
Syntax: datatype Arrayname[size];	Syntax: datatype Arrayname[rowsize][col-size];
Example: int A[10];	Example: int A[10][5];

3. **Explain (i) Linear search method (ii) Binary search method. Which of the two is more efficient for sorted data?**

Ans. (i) Linear search method: In linear search, each element of the array is compared with the given Item to be searched for, one by one.

(ii) Binary search method: Binary search technique searches the given Item in a sorted array. The search segment reduces to half at every successive stage.

The binary search method is more efficient for sorted data.

4. **Explain sorting along with two popular sort techniques.**

Ans. Sorting of an array means arranging the array elements in a specified order i.e., either ascending or descending order. Two popular sort techniques are as following:

(i) Selection Sort: In selection sort, the smallest key from the remaining unsorted array is searched for and put in the sorted array. This process repeats until the entire array is sorted.

(ii) Bubble Sort: In bubble sort, the adjoining values are compared and exchanged if they are not in proper order. This process is repeated until the entire array is sorted.

5. **Design an algorithm that will allow you to insert a data item NAM in the ith position in a single-dimensional array NAMES having an element (i<n).**

Ans.

```
1.[Initialize the value of ctr] Set ctr=n
2.Repeat for ctr=n down to pos[Shift the elements down by one position]
  Set NAMES[ctr+1]=NAMES[ctr][End of loop]
3.[insert the elements at required position]Set NAMES[i]=NAM
4.[Reset n] Set n= n+1
5.Display the new list of arrays
6.End
```

6. **Write a user-define function Upper-half() which takes a two dimensional array A, with N rows and N columns as argument and point the upper half of the array.**

```
                  2 3 1 5 0                                     2 3 1 5 0
                  7 1 5 3 1                                       1 5 3 1
    e.g., If A is 2 5 7 8 1          The output will be             7 8 1
                  0 1 5 0 1                                           0 1
                  3 4 9 1 5                                             5
```

Ans.

```
Void Upper-half(int b[][10],int N)
{     int i,j;
      for(i=N;i>0;i--)
      {     for(j=N;j>0;j--)
            {     if(i>=j)
                        cout<<b[i][j]<<" ";
```

```
                    else
                          cout<<" ";
              }
              cout<<"\n";
        }
}
```

7. **Consider the linear array A[-10;10], B[1925:1990], C[25].**
(a) Find the number of elements in each array
(b) Suppose base (A)=400 and word size (A)=2 words, find the addresses of A[-3], A[0], A[3]

Ans. (a) U-L+1 = 10-(-10)+1=21, U-L+1 = 1990-1925+1=66, 25

(b) Address of A[-3] = 400 + 2(-3-(-3))
= 400+2(0)
= 400

Address of A[0] = 400 + 2(0-(-3))
= 400+2(3)
= 406

Address of A[3] = 400 + 2(3-(-3))
= 400+2(6)
= 412

8. **Suppose an array A[10] stores numeric values only. Write algorithms to**
(i) calculate the average of the values in A
(ii) print the even numbers stored in A.

Ans. **(i) calculate the average of the values in A**

```
1. Start
2. read A[i]
3. let i=0,sum=0,avg=0
4. add A[i] into sum and store it into sum
5. divide sum by 10 and store it in avg
6. check whether i is less than 10 if yes than increment i by 1
7. print avg
8. End
```

(ii) print the even numbers stored in A.

```
1. Start
2. read A[i]
3. let i=0
4. check whether A[i]%2 is equal to 0
5. if yes than print A[i]
6. check whether i is less than 10 if yes than increment i by 1
7. End
```

9. **An array Arr[50][100] is stored in the memory along the row with each element occupying 2 bytes of memory. Find out the base address of the location Arr[20][50], if the location of Arr[10][25] is stored at the address 10000.**

Ans.

```
Address of Arr[i][j] along the row= Base Address + w*(I*C+j)
            Address of Arr[10][25]= Base Address + 2*(10*100+25)
                             10000= Base Address + 2*(1025)
                             10000= Base Address + 2050
                      Base Address= 10000-2050
                      Base Address= 7950

Address of Arr[20][50]= Base Address + 2*(20*100+50)
```

```
                                   = 7950 + 2*(2050)
                                   = 7950 + 4100
                                   = 12050
```

10. **An array Array[20][15] is stored in the memory along the column with each element occupying 8 bytes of memory. Find out the Base address and address of the element Array [2][3], if the element Array [10][25] is stored at the address 1000.**

Ans.
Given A[R][C]=A[20][15]
i.e., R=20
C=15
Element size W=8 bytes
Base Address B=?
Lowest row index lr=0, Lowest column index lc=0
Given A[4][5]=1000
To find address of A[2][3]
Address in column major is calculated as
A[I][J]=B+W(I-lr)+R(J-lc)
Since we have A[4][5]=1000, we get
1000=B+8((4-0)+20(5-0))
1000=B+832
Base Address B=1000-832=168
Now A[2][3]=B+W(I-lr)+R(J-lc)
=168+8((2-0)+20(3-0))
=168+469
A[2][3]=664

11. **An array X[7][20] is stored in the memory with each element requiring 2 bytes of storage. If the base address of array is 2000, calculate the location of X[3][5] when the array X is stored in column major order.**
Note: X[7][20] means valid row indices are 0 to 6 and valid column indices are 0 to 10.

Ans.
Address in column major is calculated as
X[I][J]=B+W(I-lr)+R(J-lc)
X[3][5]=2000+2((3-0)+7(5-0))
=2000+2(3+35)
=2000+2(38)
=2000+76
=2076

12. **An array Arr[15][20] is stored in the memory along the row with each element occupying 4 bytes of memory. Find out the Base address and address of the element Arr[3][2], if the element Arr[10][25] is stored at the address 1500.**

Ans.

```
Total no. of Rows R=15
Total no. of Columns C=20
Lowest Row lr=0
Lowest Column lc=0
Size of element W=4 bytes
Arr[I][J]  i.e., Arr[5][2]=1500
Arragement Order:Row wise
Base Address B=?
=> Arr[I][J]=B+W(C(I-lr)+(J-lc))
   Arr[5][2]=B+4(20(5-0)+(2-0))
        1500=B+408
   B=1092
Base Address=1092
```

	Arr[3][2]=B+W(C(3-0)+(2-0)) =1092+4(20(3-0)+(2-0)) =1092+248 =1340 Arr[3][2]=1340
13.	**An array X[10][20] is stored in the memory with each element requiring 4 bytes of storage. If the base address of array is 1000, calculate the location of X[5][15] when the array X is stored in column major order.** **Note: X[10][20] means valid row indices are 0 to 9 and valid column indices are 0 to 19.**
Ans.	Address in column major is calculated as X[I][J]=B+W(I-lr)+R(J-lc) X[5][15]=1000+4((5-0)+10(15-0)) =1000+4(5+150) =1000+4(155) =1000+620 =1620
14.	**An array VAL[1..15][1..10] is stored I the memory with each element requiring 4 bytes of storage. If the base address of array VAL is 1500, determine the location of VAL[12][9] when the array VAL is stored (i) Row wise (ii) Column wise.**
Ans.	Base address B=1500 Element width w=4 bytes Total rows r=15 Total columns c=10 ARR[I][J] = ARR[12][9] => I=12, J=9 Lowest row index l_r=0 in C++ Lowest column index l_c=0 in C++ (i) Row wise VAL[I][J] = B + w(c(I – l_r) + (J – L_c)) VAL[12][9] = 1500 + 4(10(12-1) + (9-1)) = 1500 + 472 = 1972 (ii) Column wise VAL[I][J]=B+W(I-lr)+R(J-lc) =1500+4((12-1)+15(9-1)) =1500+4(131) =1500+524 =2024
15.	**An array ARR[5][5] is stored in the memory with each element occupying 4 bytes of space. Assuming the base address of ARR to be 1000, compute the address of ARR[2][4], when the array is stored:** **(i) Row wise (ii) Column wise.**
Ans.	Base address B=1500 Element width w=4 bytes Total rows r=5 Total columns c=5 ARR[I][J] = ARR[2][4] => I=2, J=4 Lowest row index l_r=0 in C++

	Lowest column index l_c=0 in C++ (i) Row wise ARR[I][J] = B + w(c(J – l_r) + (J – L_c)) ARR[2][4] = 1000 + 4(5(2-0) + (4-0)) = 1000 + 56 = 1056 (ii) Column wise ARR[2][4] = B + w((I – l_r) + r(I – l_c)) ARR[2][4] = 1000 + 4((2-0) + 5(4-0)) = 1000 + 88 = 1088
16.	**Each element of an array DATA[1..10][1..10] requires 8 bytes of storage. If base address of array DATA is 2000, determine the location of DATA[4][5], when the array is stored (i) Row-wise (ii) Column-wise.**
Ans.	Base address B=2000 Element width w=8 bytes Total rows r=10 Total columns c=10 ARR[I][J] = ARR[4][5] => I=4, J=5 Lowest row index l_r=0 in C++ Lowest column index l_c=0 in C++ (i) Row wise DATA[I][J] = B + w(c(I – l_r) + (J – L_c)) DATA [4][5] = 2000 + 8(10(4-1) + (5-1)) = 2000 + 272 = 2272 (ii) Column wise DATA [I][J]=B+W(I-lr)+R(J-lc) =2000+8((4-1)+10(5-1)) =2000+8(43) =2000+344 =2344
17.	**An array Arr[40][30] is stored in the memory along the column with each element occupying 4 bytes. Find out the base address and address of the element S [22][15], if the element S[15][10] is stored at the memory location 7200.**
Ans.	Address of S[i][j] along the column =Base Address + W [(i-L1) + (j-L2) * M] Address of S[15][10] = Base Address + 4 [(15 - 1) + (10-1) x 40] 7200 = Base Address + 4 [374] Base Address = 7200 - (4 X 374) Base Address = 7200 - 1496 = 5704 Address of S[20][15] = 5704 + 4 ((20 - 1) + (15 - 1) x 40) = 5704 + 4 x 579 = 5704 + 2316 = 8020

18.	**An array T[50][20] is stored in the memory along the column with each element occupying 4 bytes. Find out the base address and address of the element T [30][15], if the element T[25][10] is stored at the memory location 9800.**
Ans.	T[50][20] No. of Rows(i.e., R) = 50 No. of Cols(i.e., C) = 20 Element size(W) = 4 bytes T[I][J] = T[30][15] => I=30, J=15 Address of T[25][10] = 9800 Base Address (*B*) =? Lowest Row (i.e., l_r) = 0 Lowest Col (i.e., l_c) = 0 Formula to calculate address in Column Major arrangement is: T[P][Q] = B + W[(*P* - l_r) + *R*(*Q* - l_c)] T[25][10] = B + 4((25 – 0) + 50(10 – 0)) 9800 = B + 4(525) (∵ T[25][10] = 9800 given) 9800 = B + 2100 => B = 9800 – 2100 = 7700 Parallely, T[I][J] = B + W[(*I* - l_r) + *R*(*J* - l_c)] T[30][15] = 7700 + 4[(30 – 0) + 50(15 – 0)] = 7700 + (4 x 780) = 7700 + 3120 = 10820
19.	**An array T[90][100] is stored in the memory along the column with each element occupying 4 bytes. Find out the memory location for the element T [10][40], if the Base Address of the array is 7200.**
Ans.	Loc(T[I][J)) = Base(T)+W(I+J*N) (where N is the number of rows, LBR = LBC = 0) = 7200 + 4[10 + 40 x 90] = 7200 + 4[10+3600] = 7200 + 4 x 3610 = 7200 + 14440 = 21640
20.	**An array Arr[35][15] is stored in the memory along the row with each element occupying 4 bytes. Find out the base address and address of an element Arr[20][5], if the location Arr[2][2] is stored at the address 3000.**
Ans.	A[35][15] => rows R=35, columns C=15 Let base address be B Given element width W=4 bytes and A[2][2]=3000 In Row major, A[I][J]=B+W(C(I=lr)+(j-lc)) where lr=lowest row and lc=lowest column A[2][2]=B+W(C(2-0)+(2-0)) 3000=B+4(15(2)+2) 3000=B+128 Base Address B=3000-128=2872 Using same formula A[20][5]=2872+4(15(20-0)+(5-0)) =2872+1220 =4092
21.	**An array Arr[15][35] is stored in the memory along the column with each element occupying 8 bytes. Find out**

the base address and address of the element Arr[2][5], if the location Arr[5][10] is stored at the address 4000.

Ans.

Arr[15][35]
No. of Rows(i.e., R) = 15
No. of Cols(i.e., C) = 35
Element size(W) = 8 bytes
Arr[I][J] = Arr[2][5] => I=2, J=5

Address of Arr[5][10] = 4000
Base Address (*B*) =?
Lowest Row (i.e., l_r) = 0
Lowest Col (i.e., l_c) = 0

Formula to calculate address in Column Major arrangement is:

Arr[P][Q] = B + W[(*P* - l_r *)* + *R(Q* - l_c*)*]
Arr[5][10] = B + 8((5 – 0) + 15(10 – 0))
4000 = B + 8(155)
4000 = B + 1240
=> B = 4000-1240= 2760
Parallely, Arr[I][J] = B + W[(*I* - l_r *)* + *R(J* - l_c*)*]
Arr[2][5] = 2760 + 8[(2 – 0) + 15(5 – 0)]
= 2760 + (8 x 77)
= 2760 + 616
= 3376

22. **An array MAT[30][10] is stored in the memory column wise with each element occupying 8 bytes of memory. Find out the base address and the address of element MAT[20][5], if the location of MAT[5][7] is stored at the address 1000.**

Ans.

Base Address B
No of rows m=30
Element size W=8
Lowest Row and column indices l_r, l_c=0
Address of Ith, jth element of array in column major order is:
Address of MAT[I][J] = B + W(m(J - l_c) + (I - l_r))
MAT[5][7] = 1000
1000 = B + 8(30(7-0)+(5-0))
1000 = B + 8(30(7)+(5))
1000 = B + 8(210 + 5)
1000 = B + 8(215)
1000 = B + 1720
B = 1000 – 1720
B = -720

⇨ Base address is -720

Now address of MAT[20][5] is computed as:
MAT[20][5] = -720 + 8(30(5 – 0) + (20 – 0))
= -720 + 8(30(5) + (20))
= -720 + 8(150 + 20)
= -720 + 8(170)
= -720 + 1360
= 640

23 **Write an algorithm to search for an ITEM linearly in array X[-10:10]**

Ans.

```
1. ctr=-10
2. Repeat steps 3 and 4 until ctr>10
```

```
3.    If X[ctr]==ITEM then
      {    print "Search Successfull"
           Print ctr, “is the location of”, ITEM
           break
      }
4. ctr=ctr+1
   /* End of Repeat */
5. if ctr>10 then
      print "Search Unsuccessfull"
6. END
```

24. **Write an algorithm to search for an ITEM using binary search in array X[-10:10]**

Ans.

```
1. Set beg=-10,last=10
2. REPEAT steps 3 through 6 UNTIL beg>last   //INT() is used to extract
integer part
3.    mid=INT((beg+last)/2)
4.    if X[mid]==ITEM then
      {    print "Search  Successful"
           print ITEM,"fount at",mid
           break                    /* go out of the loop*/
      }
5.    if X[mid]<ITEM then
           beg=mid+1
6.    if X[mid]>ITEM then
           last=mid-1
      /* END of repeat */
7. if beg!=last
      print "Unsuccessful Search"
8. END
```

25. **Write an algorithm to insert ITEM at an appropriate position in array X[-10:10]**

Ans.

```
1. ctr=-10                 /*Initialize the counter */
2. If LST=10 then
   { print "Overflow:"
      Exit from program
   }
3. if X[ctr]>ITEM then
      pos=1
   else
   {
4.    Repeat steps 5 and 6 until ctr>=10
5.    if X[ctr]<=ITEM and ITEM<=X[ctr+1] then
      {    pos=ctr+1
           break
      }
6.    ctr=ctr+1
7.    ctr=10 then
          pos=10+1
   }                            /* end of if step 3*/
   /* shift the elemets to create space */
8. ctr=10                       /*Initialize the counter */
9. while ctr>=pos perform steps 10 through 11
10. { X[ctr+1]=X[ctr]
11.   ctr=ctr-1
    }
12. X[pos]=ITEM                 /* Insert the elements */
13. END.
```

26. **Write an algorithm to delete an ITEM at position 0 in array X[-3:5]. The free space should be put in the beginning of array.**

Ans.

```
1. ctr=-3
2. If LST=0
   { print "Underflow"
     Exit from program
   }
3. If(pos==0)
     X[pos]=0;
     print "Zero (0) signifies the deleted element"
/*After this the free space will be put in the beginning of array */
4. ctr=pos
5. Repeat steps 6 and 7 until ctr>=5
6.   X[ctr]=X[ctr-1]
7.   ctr=ctr-1
/* End of Repeat*/
8. END
```

27. **Write algorithm to sort an array B[-3:5]:**
(i) using selection sort (ii) using bubble sort

Ans. **(i) using selection sort:**

```
1. L=-3, U=5
2. small=B[L]         /* Initialize small with first array element */
3. For I=L TO U do
   {
4. small=B[I],pos=I
   /* Loop to find smallest element and its positoon */
5.    For J=I TO U do
      {
6.    If B[J]<small then
7.    {    small=B[J]
8.         pos=J
      }
      J=J+1
   }            /*end of inner loop*/
   /* swap the smallest element with I^th element*/
9.  temp=B[I]
10. B[I]=small
11. B[pos]=temp
   }       /*end of outer loop*/
12. END.
```

(ii) using bubble sort:

```
1. L=-3, U=5
2. For I=L TO U
3. { For J=L TO [(U-1)-I]  //need not consider already settled heavy
elements//
                             // that is why (U-1)-I
4.    { if B[J]>B[J+1] then
         {             /* swap the values*/
5.         temp=B[J]
6.         B[J]=B[J+1]
7.         B[J+1]=temp
         }             /*end of if*/
      }                /*end of inner loop*/
```

```
    }                   /*end of outer loop*/
8. END.
```

28. **The following array of integers is to be arranged in ascending order using the bubble sort technique:**
26 21 20 23 29 17
Give the contents of array at the end of each iteration. Do not write the algorithm.

Ans. Bubble Sort (Bold elements depicts that they are to be compared in the next pass.)

Step 1. 26, 21, 20, 23, 29, 17
Step 2. 21, 26, 20, 23, 29, 17
Step 3. 21, 20, 26, 23, 29, 17
Step 4. 21, 20, 23, 26, 29, 17
Step 5. 21, 20, 23, 26, 29, 17
Step 6. 21, 20, 23, 26, 17, 29
Step 7. 20, 21, 23, 26, 17, 29
Step 8. 20, 21, 23, 26, 17, 29
Step 9. 20, 21, 23, 26, 17, 29
Step 10. 20, 21, 23, 17, 26, 29
Step 11. 20, 21, 23, 17, 26, 29
Step 12. 20, 21, 23, 17, 26, 29
Step 13. 20, 21, 23, 17, 26, 29
Step 14. 20, 21, 17, 23, 26, 29
Step 15. 20, 21, 17, 23, 26, 29
Step 16. 20, 21, 17, 23, 26, 29
Step 17. 20, 21, 17, 23, 26, 29
Step 18. 20, 17, 21, 23, 26, 29
Step 19. 20, 17, 21, 23, 26, 29
Step 20. 20, 17, 21, 23, 26, 29
Step 21. 20, 17, 21, 23, 26, 29
Step 22. 17, 20, 21, 23, 26, 29

29. **Write an algorithm to merge two arrays X[6], Y[5] stored in descending order. The resultant array should be in ascending order.**

Ans.

```
Assuming that L=0 and U=6-1,5-1 and (6+5)-1 respectively for X, Y, and Z
1.  ctrX=6-1; ctrY=5-1; ctrZ=0;
2.  while ctrX>=0 and ctrY>=0 perform steps 3 through 10
3.  { If X[ctrX]<=Y[ctrY] then
4.    {    Z[ctrZ]=X[ctrX]
5.         ctrZ=ctrZ+1
6.         ctrX=ctrX-1     }
7.      else
8.      {  Z[ctrZ]=Y[ctrY]
9.         ctrZ=ctrZ+1
10.        ctrY=ctrY-1      }
    }
11.  if ctrX<0 then
12.  {    while ctrY>=0 perform steps 13 through 15
          {
13.           Z[ctrZ]=Y[ctrY]
14.           ctrZ=ctrZ+1
15.           ctrY=ctrY-1
          }
     }
16.  if ctrY<0 then
17.  {   while ctrX>=0 perform steps 18 through 20
18.      {    Z[ctrZ]=X[ctrX]
19.            ctrZ=ctrZ+1
20.            ctrX=ctrX-1
         }
     }
```

30. **Write an algorithm to add corresponding elements of two matrices A[3 x 3] and B[3 x 3]**

Ans.

```
/* Read the two matrices */
1.  for i=1 to 3 perform step 2 through 4
2.  {  for j=1 to 3 perform step 3 through 4
3.     {     Read A[i,j]
4.           Read B[i,j]
     }
    }
/* Calculate the sum of the two and store it in third matrix C */
5.  for i=1 to 3 perform step 6 through 8
    {
6.      for j=1 to 3 perform step 7 through 8
7.      {   C[i,j]=0
8.          C[i,j]=A[i,j]+B[i,j]
     }
    }
```

31. **Write an algorithm to subtract a matrix A[4 x 4] from a matrix X[4 x 4]**

Ans.

```
/* Read the two matrices */
1.  for i=1 to 4 perform step 2 through 4
2.  {  for j=1 to 4 perform step 3 through 4
3.     {     Read A[i,j]
4.           Read B[i,j]
     }
    }
/* Calculate the sum of the two and store it in third matrix C */
5.  for i=1 to 4 perform step 6 through 8
    {
6.      for j=1 to 4 perform step 7 through 8
7.      {   C[i,j]=0
8.          C[i,j]=A[i,j]-B[i,j]
     }
    }
```

32. **Write an algorithm to print all those elements of a matrix X[4 x 4] that are not diagonal elements.**

Ans. Students I am giving you the program for printing Non Diagonal elements of a matrix X[4x4], try to convert this code into algorithm.

```
#include<conio.h>
#include<iostream.h>
  void accept(int a[4][4],int size)
  {
    cout<<"Diagonal One:";
    for (int i=0;i<size;i++)
      for(int j=0;j<size;j++)
        if (i != j && i != size-j-1)
           cout<<a[i][j];
}
void main()
{
    int a[4][4]={{5,4,3,4},{6,7,9,1},{8,0,3,7},{2,4,5,9}};
    clrscr();
    accept(a,4);
    getch();
}
```

33. **Write a user-defined function in C++ to find and display the sum of both the diagonal elements of a two dimensional array MATRIX[6][6] containing integers.**

Ans.

```
float diagonalSum(float MATRIX[6][6], int r, int c)
{
     int i,j;
     float sum=0;
     //We are calculating sum of diagonal elements considering both diagonals
     //We are adding intersecting element on two diagonal twice
     for(i=0;i<r;i++)
     {
          for(j=0;j<c;j++)
          {
               if(i==j)              //elements on first diagonal
               sum+= MATRIX [i][j];
               if((i+j)==(r-1))      // elements on off-diagonal
               sum+= MATRIX [i][j];
          }
     }
     return sum;
}
```

34. **What is the pre-condition for applying binary search algorithm?**

Ans. For applying binary search algorithm the array, to be scanned, must be sorted in any order (ascending or descending).

35. **Write a user defined function in C++ to display the multiplication of row elements of two dimensional array A[4][6] containing integers.**

Ans.

```
void RowMulti(int A[4][6])
{    int Mul[4];
     for(int i=0;i<4;i++)
     {    Mul[i]=1;
          for(int j=0;j<6;j++)
               Mul[i]*=A[i][j];
          cout<<"Product of row"<<i+1<<"="<<Mul[i]<<endl;
     }
}
```

36. **Write a user defined function in C++ to display the sum of row elements of two dimensional array A[5][6] containing integers.**

Ans.

```
void RowSum(int A[5][6])
{    int SUMC[5];
     for(int i=0;i<5;i++)
     {    SUMC[i]=0;
          for(int j=0;j<6;j++)
               SUMC[i]+=A[i][j];
          cout<<"Sum of row"<<i+1<<"="<< SUMC[i]<<endl;
     }
}
```

37. **Write a user defined function in C++ to display the sum of column elements of two dimensional array R[7][7] containing integers.**

Ans.

```
void COLSUM(int R[7][7])
{
     int SUMC;
     for (int j=0;j<7;j++)
     {
       SUMC=0;
       for(int i=0;i<7;i++)
          SUMC=SUMC + R[i][j];
       Cout<< "Sum of Column "<<j<<" = "<<SUMC ;
     }
}
```

38. **Write a user defined function in C++ to display the sum of row elements of two dimensional array M[5][6] containing integers.**

Ans. Same as Q. No. 36

39. **Consider the following key set: 42, 29, 74, 11, 65, 58, use insertion sort to sort the data in ascending order and indicate the sequences of steps required.**

Ans. Insertion sort:

Step-1 - ∞, 42, 29, 74, 11, 65, 58
Step-2 - ∞, 29, 42, 74, 11, 65, 58
Step-2 - ∞, 29, 42, 11, 74, 65, 58
Step-4 - ∞, 29, 42, 11, 65, 74, 58
Step-5 - ∞, 29, 42, 11, 58, 65, 74
Step-6 - ∞, 11, 29, 42, 58, 65, 74

40. **Write a user-defined function in C++ to display those elements of a two dimensional array T[4][4] which are divisible by 100. Assume the content of the array is already present and the function prototype is as follows:**

`void Showhundred(int T[4][4]);`

Ans.

```
void Showhundred(int T[4][4])
{
       for(int I = 0; I<4; I++)
        {
          for(int J = 0; J<4; J++)
            {
                if (T[I][J]%100 = = 0)
                     cout<<"Elemets which are divisible by 100 are:”
                         <<A[I][J]<<endl;
             }
        }
}
```

41. **Each element of two-dimensional array (with 5 rows and 4 columns) is stored in one memory location. If A(1,1) is at location 2000, what is the address of A(4,4) ? The arrangement is row-major. Use a suitable formula for the calculation.**

Ans. A[5][4] => rows R=5, columns C=4

Let base address be B

Given element width W=1 bytes and A[1][1]=2000

In Row major,

A[I][J]=B+W(C(I=lr)+(j-lc))

where lr=lowest row and lc=lowest column

A[1][1]=B+W(C(1-0)+(1-0))
2000=B+1(4(1)+1)
2000=B+5

Base Address B=2000-5=1995

Using same formula

A[4][4]=1995+1(4(4-0)+(4-0))
=1995+20

```
                        =2015
```

42. **Calculate the address of X[4,3] in a two-dimensional array X[1….5, 1…..4] stored in row=major order in the main memory. Assuming base address to be 1000 and that each requires 4 words of storage.**

Ans.

```
     X[4][3]=B+W(C(I-1)+(J-1))
            =1000+4(4(4-1)+(3-1))
            =1000+4(4(3)+(2))
            =1000+56
            =1056
```

LONG ANSWER QUESTIONS

1. **What are data structures? What are their types and sub-types? Explain each of the subtypes with examples.**

Ans. The data structures are named group of data of some data types. The data structures can be classified into following two types:

1. Simple Data Structure: These data structure are normally built from primitive data types like integers, reals, characters, boolean. Simple data structure can be classified into following two categories:

(a) Array: Arrays refer to a named list of a finite number n of similar data elements.

For example,   int ARR[10];

Above array ARR have 10 elements, each elements will be referenced as Arr[0], ARR[1]…………ARR[9].

(b) Structure: Structure refers to a named collection of variables of different data types.

For example, a structure named as STUD contais (Rno, Name, Mark), then individual fields will be referenced as STUD.fieldname such as, STUD.Rno, STUD.Name etc.

2. Compound Data Structure: Simple data structure can be combine in various waus to form more complex structures called compound data structures which are classified into following two categories:

(a) Linear data structure: These data structures are a single level data structures representing linear relationship among data. Following are the types of linear data structure:

(i) Stacks: Stack is a LIFO (Last In First Out) list. For example, stack of plates on counter, as that plates are inserted or removed only from the top of the stack.

(ii) Queue: Queue is a FIFO (First In First Out) list. For example, line of people waiting for their turn to vote.

(iii) Linked List: Linked lists are special lists of some data elements liked to one another. For example, peoples seating in a cinema hall where each seat is connected to other seat.

(b) Non-Linear data structure: These data structures are multilevel data structure representing hierarchical relationship among data. For example, relationship of child, parent and grandparent.

2. **Write an algorithm to search for given ITEM in a given array X[n] using linear search technique. If the ITEM is found, move it at the top of the array. If the ITEM is not found, insert it at the end of the array.**

Ans. Students I gave you solution of 2 part of the question

First part Linear Search Technique Algorithm

```
1. LB=0
2. Repeat steps 3 and 4 until LB>UB      //UB means Upper Bound(length of array)
3. If ARR[LB]==ITEM then
    {
      pos=LB
      break
    }
4. LB=LB+1
5. if LB>UB then
print "Value Not Found"
else
{                      //Second part swapping of searched item at top of the array
temp=ARR[pos]
ARR[pos]=ARR[0]
ARR[0]=temp
}
```

Third part is inserting the item which is not present at the end of the array, try this part.

3. **Write an algorithm to search for 66 and 71 in the following array:**

3, 4, 7, 11, 18, 29, 45, 71, 87, 89, 93, 96, 99

Make use of binary search technique. Also give the intermediate results while executing this algorithm. Convert this algorithm into a C++ program.

Ans. **Algorithm:**

```
1. Set beg=0,last=12
2. REPEAT steps 3 through 6 UNTIL beg>last   //INT() is used to extract
integer part
3.    mid=INT((beg+last)/2)
4.    if A[mid]==ITEM then
      {     print "Search  Successful"
            print ITEM,"fount at",mid
            break                        /* go out of the loop*/
      }
5.    if A[mid]<ITEM then
            beg=mid+1
6.    if A[mid]>ITEM then
            last=mid-1
      /* END of repeat */
7. if beg!=last
      print "Unsuccessful Search"
8. END
```

Intermediate Results:

```
(i) Search for 66.
Step 1:
      beg=1; last=13; mid=INT(1+13)/2=7
Step 2:
      A[mid] i.e., A[7] is 45
      45<66 then
      beg=mid+1 i.e., beg=7+1=8
Step 3:
      mid=Int((beg+last)/2)=INT((8+13)/2)=10
      A[10] i.e., 89>66 then last = mid-1=10-1=9
Step 4:
      mid=((8+9)/2)=8
      A[8] is 71
       71>66 than last = mid-1=8-1=7
Step 5:
      mid=((8+7)/2)=7
      A[7] is 45
      45 < 66 then beg = mid+1=7+1=8
Step 6:
      mid=((8+8)/2)=8   (beg=last=8)
      A[8] is 71   => 71!=66
“Search Unsuccessful!!!”
(ii) Search for 71.
Step 1:
      beg=1; last=13; mid=INT(1+13)/2=7
Step 2:
      A[mid] i.e., A[7] is 45
      45<71 then
      beg=mid+1 i.e., beg=7+1=8
Step 3:
      mid=Int((beg+last)/2)=INT((8+13)/2)=10
      A[10] i.e., 89>71 then last = mid-1=10-1=9
Step 4:
      mid=((8+9)/2)=8
      A[8] is 71   71=>71
“Search Successful!!!”
```

Program:

```
#include<iostream.h>
int Bsearch(int [],int);
int main()
{    int A[]={3,4,7,11,18,29,45,71,87,89,93,96,99};
     int index;
     index=Bsearch(A,71);
     if(index==-1)
          cout<<"Element not found..";
     else
          cout<<"Element found at 
index:"<<index<<"/Position:"<<index+1<<endl;
     return 0;
}
int Bsearch(int A[],int item)
{    int beg,last,mid;
     beg=0; last=13-1;
     while(beg<=last)
     {    mid=(beg+last)/2;
          if(item==A[mid]) return mid;
          else if (item>A[mid]) beg=mid+1;
          else  last=mid-1;
     }
     rerurn -1;
}
```

4. **An array X[n] stores names only. The first position of the array does not store a name rather it stores the number of available free spaces in the array. Write an algorithm to insert or delete an ITEM (accept it from the users) in the given array.**

Ans. **Insert an ITEM:**

```
1. ctr=0                   /*Initialize the counter */
2. If LST=n then
   { print "Overflow:"
     Exit from program
   }
3. if X[ctr]>ITEM then
     pos=1
   else
   {
4.   Repeat steps 5 and 6 until ctr>=U
5.   if X[ctr]<=ITEM and ITEM<=X[ctr+1] then
     {    pos=ctr+1
          break
     }
6.   ctr=ctr+1
7.   ctr=n then
         pos=n+1
   }                             /* end of if step 3*/
   /* shift the elemets to create space */
```

```
8. ctr=10                        /*Initialize the counter */
9. while ctr>=pos perform steps 10 through 11
10. { X[ctr+1]=X[ctr]
11.   ctr=ctr-1
    }
12. X[pos]=ITEM                  /* Insert the elements */
13. END.
```

Delete an ITEM

```
1. ctr=0
2. If n=0
   { print "Underflow"
     Exit from program
   }
3. Repeat steps 4 and 5 until ctr<n
4.   if(X[ctr]==ITEM) return ctr
     return -1
5. if(pos!=-1)
     X[pos]=0;
     print "Zero (0) signifies the deleted element"
/*After this the free space will be put in the end of array */
6. ctr=pos
7. Repeat steps 6 and 7 until ctr>=5
8.   X[ctr]=X[ctr+1]
9.   ctr=ctr+1
/* End of Repeat*/
10.END
```

5. **In array A[n], after deletion of ay element, no element was shifted, thus, the free space is scattered across the array. You have been given the task to solve this problem. Write an algorithm to combine all the elements at the rear end of the array so that all the free spaces are available at the beginning of the array.**

Ans.

```
1.ctr=pos
2.Repeat steps 3 and 4 until ctr<=1
3.   A[ctr]=A[ctr-1]
4.   ctr=ctr-1
/* End of Repeat*/
5.Display the new list of element
6.End
```

6. **Given the following array:**

13, 7, 6, 21, 35, 2, 28, 64, 45, 3, 5, 1

Write an algorithm to sort the above array using exchange selection sort. Give the array after every iteration. Convert this algorithm into C++ program.

Ans. **Algorithm:**

```
1. L=0, U=11
2. small=A[L]          /* Initialize small with first array element */
3. For I=L TO U do
   {
4. small=A[I],pos=I
   /* Loop to find smallest element and its position */
5.    For J=I TO U do
      {
6.   If A[J]<small then
7.   {    small=A[J]
8.        pos=J
     }
     J=J+1
   }            /*end of inner loop*/
   /* swap the smallest element with Ith element*/
9.  temp=A[I]
10. A[I]=small
11. A[pos]=temp
   }       /*end of outer loop*/
12. END.
```

Array status after every iteration:

Note: element with red color is smallest element

```
(1)  13, 7, 6, 21, 35, 2, 28, 64, 45, 3, 5, 1
(2)  1, 7, 6, 21, 35, 2, 28, 64, 45, 3, 5, 13
(3)  1, 2, 6, 21, 35, 7, 28, 64, 45, 3, 5, 13
(4)  1, 2, 3, 21, 35, 7, 28, 64, 45, 6, 5, 13
(5)  1, 2, 3, 5, 35, 7, 28, 64, 45, 6, 21, 13
(6)  1, 2, 3, 5, 6, 7, 28, 64, 45, 35, 21, 13
(7)  1, 2, 3, 5, 6, 7, 13, 64, 45, 35, 21, 28
(8)  1, 2, 3, 5, 6, 7, 13, 21, 45, 35, 64, 28
(9)  1, 2, 3, 5, 6, 7, 13, 21, 28, 35, 64, 45
(10) 1, 2, 3, 5, 6, 7, 13, 21, 28, 35, 45, 64
```

Program:

```
#include<iostream.h>
void SelSort(int []);
int main()
{     int A[]={13,7,6,21,35,2,28,64,45,3,5,1};
      SelSort(A);
      cout<<"The sorted array is as following...";
      for(i=0;i<12;i++)
            cout<<A[i]<<" ";
      cout<<endl;
      return 0;
}
void SelSort(int A[])
{     int small,pos,tmp;
      for(int i=0;i<12;i++)
      {     small=A[i]'
            pos=i;
            for(int j=i+1;j<size;j++)
            {     if(A[j]<small)
                  {  small=A[j];  pos=j; }
            }
            tmp=A[i];
            A[i]=A[pos];
            A[pos]=tmp;
            cout<<"\n Array after pass-"<,i+1<<"-is:";
            for(j=0;j<size;j++) cout<<A[j]<<" ";
      }
}
```

7. **For the same array mentioned above in question 6, write an algorithm to sort the above array using bubble sort technique. Give the array-status after every iteration.**

Ans. **Algorithm:**

```
1. L=0, U=11
2. For I=L TO U
3. { For J=L TO [(U-1)-I]  //need not consider already settled heavy
elements//
                             // that is why (U-1)-I
4.    { if A[J]>A[J+1] then
         {             /* swap the values*/
5.         temp=A[J]
6.         A[J]=A[J+1]
7.         A[J+1]=temp
         }             /*end of if*/
      }                /*end of inner loop*/
    }                  /*end of outer loop*/
8. END.
```

Array status after every iteration:

Note: Element in red color depict that they are to be compared in the next pass.

```
(1)  13, 7, 6, 21, 35, 2, 28, 64, 45, 3, 5, 1
(2)  7, 13, 6, 21, 35, 2, 28, 64, 45, 3, 5, 1
(3)  7, 6, 13, 21, 35, 2, 28, 64, 45, 3, 5, 1
(4)  7, 6, 13, 21, 35, 2, 28, 64, 45, 3, 5, 1
(5)  7, 6, 13, 21, 35, 2, 28, 64, 45, 3, 5, 1
(6)  7, 6, 13, 21, 2, 35, 28, 64, 45, 3, 5, 1
(7)  7, 6, 13, 21, 2, 28, 35, 64, 45, 3, 5, 1
(8)  7, 6, 13, 21, 2, 28, 35, 64, 45, 3, 5, 1
(9)  7, 6, 13, 21, 2, 28, 35, 45, 64, 3, 5, 1
(10) 7, 6, 13, 21, 2, 28, 35, 45, 3, 64, 5, 1
(11) 7, 6, 13, 21, 2, 28, 35, 45, 3, 5, 65, 1
(12) 7, 6, 13, 21, 2, 28, 35, 45, 3, 5, 1, 65
//(13) 6, 7, 13, 21, 2, 28, 35, 45, 3, 5, 1, 65
//(14) 6, 7, 13, 21, 2, 28, 35, 45, 3, 5, 1, 65
(15) 6, 7, 13, 21, 2, 28, 35, 45, 3, 5, 1, 65
//(16) 6, 7, 13, 2, 21, 28, 35, 45, 3, 5, 1, 65
//(17) 6, 7, 13, 2, 21, 28, 35, 45, 3, 5, 1, 65
//(18) 6, 7, 13, 2, 21, 28, 35, 45, 3, 5, 1, 65
(19) 6, 7, 13, 2, 21, 28, 35, 45, 3, 5, 1, 65
(20) 6, 7, 13, 2, 21, 28, 35, 3, 45, 5, 1, 65
(21) 6, 7, 13, 2, 21, 28, 35, 3, 5, 45, 1, 65
//(22) 6, 7, 13, 2, 21, 28, 35, 3, 5, 1, 45, 65
//(24) 6, 7, 13, 2, 21, 28, 35, 3, 5, 1, 45, 65
//(25) 6, 7, 13, 2, 21, 28, 35, 3, 5, 1, 45, 65
(26) 6, 7, 13, 2, 21, 28, 35, 3, 5, 1, 45, 65
//(27) 6, 7, 2, 13, 21, 28, 35, 3, 5, 1, 45, 65
//(28) 6, 7, 2, 13, 21, 28, 35, 3, 5, 1, 45, 65
//(29) 6, 7, 2, 13, 21, 28, 35, 3, 5, 1, 45, 65
(30) 6, 7, 2, 13, 21, 28, 35, 3, 5, 1, 45, 65
(31) 6, 7, 2, 13, 21, 28, 3, 35, 5, 1, 45, 65
(32) 6, 7, 2, 13, 21, 28, 3, 5, 35, 1, 45, 65
//(33) 6, 7, 2, 13, 21, 28, 3, 5, 1, 35, 45, 65
//(34) 6, 7, 2, 13, 21, 28, 3, 5, 1, 35, 45, 65
//(35) 6, 7, 2, 13, 21, 28, 3, 5, 1, 35, 45, 65
(36) 6, 7, 2, 13, 21, 28, 3, 5, 1, 35, 45, 65
//(37) 6, 2, 7, 13, 21, 28, 3, 5, 1, 35, 45, 65
//(38) 6, 2, 7, 13, 21, 28, 3, 5, 1, 35, 45, 65
//(39) 6, 2, 7, 13, 21, 28, 3, 5, 1, 35, 45, 65
(40) 6, 2, 7, 13, 21, 28, 3, 5, 1, 35, 45, 65
(41) 6, 2, 7, 13, 21, 3, 28, 5, 1, 35, 45, 65
(42) 6, 2, 7, 13, 21, 3, 5, 28, 1, 35, 45, 65
//(43) 6, 2, 7, 13, 21, 3, 5, 1, 28, 35, 45, 65
```

	//(44) 6, 2, 7, 13, 21, 3, 5, 1, 28, 35, 45, 65 //(45) 6, 2, 7, 13, 21, 3, 5, 1, 28, 35, 45, 65 (46) 6, 2, 7, 13, 21, 3, 5, 1, 28, 35, 45, 65 //(47) 2, 6, 7, 13, 21, 3, 5, 1, 28, 35, 45, 65 //(48) 2, 6, 7, 13, 21, 3, 5, 1, 28, 35, 45, 65 //(49) 2, 6, 7, 13, 21, 3, 5, 1, 28, 35, 45, 65 (50) 2, 6, 7, 13, 21, 3, 5, 1, 28, 35, 45, 65 (51) 2, 6, 7, 13, 3, 21, 5, 1, 28, 35, 45, 65 (52) 2, 6, 7, 13, 3, 5, 21, 1, 28, 35, 45, 65 //(53) 2, 6, 7, 13, 3, 5, 1, 21, 28, 35, 45, 65 //(54) 2, 6, 7, 13, 3, 5, 1, 21, 28, 35, 45, 65 //(55) 2, 6, 7, 13, 3, 5, 1, 21, 28, 35, 45, 65 //(56) 2, 6, 7, 13, 3, 5, 1, 21, 28, 35, 45, 65 //(57) 2, 6, 7, 13, 3, 5, 1, 21, 28, 35, 45, 65 //(58) 2, 6, 7, 13, 3, 5, 1, 21, 28, 35, 45, 65 //(59) 2, 6, 7, 13, 3, 5, 1, 21, 28, 35, 45, 65 (60) 2, 6, 7, 13, 3, 5, 1, 21, 28, 35, 45, 65 (61) 2, 6, 7, 3, 13, 5, 1, 21, 28, 35, 45, 65 (62) 2, 6, 7, 3, 5, 13, 1, 21, 28, 35, 45, 65 //(63) 2, 6, 7, 3, 5, 1, 13, 21, 28, 35, 45, 65 //(64) 2, 6, 7, 3, 5, 1, 13, 21, 28, 35, 45, 65 //(65) 2, 6, 7, 3, 5, 1, 13, 21, 28, 35, 45, 65 //(66) 2, 6, 7, 3, 5, 1, 13, 21, 28, 35, 45, 65 //(67) 2, 6, 7, 3, 5, 1, 13, 21, 28, 35, 45, 65 //(68) 2, 6, 7, 3, 5, 1, 13, 21, 28, 35, 45, 65 (69) 2, 6, 7, 3, 5, 1, 13, 21, 28, 35, 45, 65 (70) 2, 6, 3, 7, 5, 1, 13, 21, 28, 35, 45, 65 (71) 2, 6, 3, 5, 7, 1, 13, 21, 28, 35, 45, 65 //(72) 2, 6, 3, 5, 1, 7, 13, 21, 28, 35, 45, 65 //(73) 2, 6, 3, 5, 1, 7, 13, 21, 28, 35, 45, 65 //(74) 2, 6, 3, 5, 1, 7, 13, 21, 28, 35, 45, 65 //(75) 2, 6, 3, 5, 1, 7, 13, 21, 28, 35, 45, 65 //(76) 2, 6, 3, 5, 1, 7, 13, 21, 28, 35, 45, 65 //(77) 2, 6, 3, 5, 1, 7, 13, 21, 28, 35, 45, 65 //(78) 2, 6, 3, 5, 1, 7, 13, 21, 28, 35, 45, 65 (79) 2, 6, 3, 5, 1, 7, 13, 21, 28, 35, 45, 65 (80) 2, 3, 6, 5, 1, 7, 13, 21, 28, 35, 45, 65 (81) 2, 3, 5, 6, 1, 7, 13, 21, 28, 35, 45, 65 //(82) 2, 3, 5, 1, 6, 7, 13, 21, 28, 35, 45, 65 //(83) 2, 3, 5, 1, 6, 7, 13, 21, 28, 35, 45, 65 //(84) 2, 3, 5, 1, 6, 7, 13, 21, 28, 35, 45, 65 (85) 2, 3, 5, 1, 6, 7, 13, 21, 28, 35, 45, 65 (86) 2, 3, 1, 5, 6, 7, 13, 21, 28, 35, 45, 65 (87) 2, 1, 3, 5, 6, 7, 13, 21, 28, 35, 45, 65 (88) 1, 2, 3, 5, 6, 7, 13, 21, 28, 35, 45, 65
8.	**Using a two-dimensional array A[n x n], write an algorithm to prepare a one-dimensional array B[n^2] that will have all the elements of A as if they are stored in column-major form.**
Ans.	Can You do this try it.
9.	**Suppose A, B, C are arrays of integers of sizes m, n, m+n respectively. The numbers in arrays A and B appear in descending order. Give an algorithm to produce a third array C, containing all the data of array A and B in ascending order.**
Ans.	Assuming that L=0 and U=m-1,n-1 and (m+n)-1 respectively for A, B, and C 1. ctrA=m-1; ctrB=n-1; ctrC=0; 2. while ctrA>=0 and ctrB>=0 perform steps 3 through 10

```
3.   { If A[ctrA]<=B[ctrB] then
4.    {    C[ctrC]=A[ctrA]
5.         ctrC=ctrC+1
6.         ctrA=ctrA-1     }
7.       else
8.       {  C[ctrC]=B[ctrB]
9.          ctrC=ctrC+1
10.         ctrB=ctrB-1      }
     }
11.   if ctrA<0 then
12.   {    while ctrB>=0 perform steps 13 through 15
           {
13.            C[ctrC]=B[ctrB]
14.            ctrC=ctrC+1
15.            ctrB=ctrB-1
           }
      }
16.   if ctrB<0 then
17.   {   while ctrA>=0 perform steps 18 through 20
18.       {    C[ctrC]=A[ctrA]
19.            ctrC=ctrC+1
20.            ctrA=ctrA-1
          }
      }
```

10. **From a two-dimensional array A[4 x 4], write an algorithm to prepare a one dimensional array B[16] that will have all the elements of A as if they are stored in row-major form. For example for the following array:**

$$\begin{pmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \\ 13 & 14 & 15 & 16 \end{pmatrix}$$

the resultant array should be

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16

Ans.

```
#include (iostream.h)

#include (conio.h)

void main( )
{
int A[4][4], B[16];

// Input elements
for(int i = 0; i < 4 ; i++)
  for( int j = 0; j < 4 ; j++)
  {
  cout<<"\n Enter elements for "<< i+1 << "," << j+1   << "location :";
  cin >> A[i][j];
 }

clrscr();
//Print the array
cout<<"\n The Original matrix : \n\n";
```

```
        for( i = 0; i < 4 ; i++)
        {
         for( j = 0; j < 4 ; j++)
            cout<< A[i][j]<<"\t";
         cout<< "\n";
        }

        int k = 0;
        // Convert 2 - D array into 1-D array by row-major rule
        for( i = 0; i < 4 ; i++)
          for( j = 0; j < 4 ; j++)
            B[k] = A[i][j];

        //display the 1D Array
        for( k=0; k<16; k++)
        cout<< B[k] << "  ";

        getch( );
        }
```

11. **Suppose a one dimensional array AR containing integers is arranged in ascending order. Write a user defined function in C++ to search for one integer from AR with the help of linear search method, returning an integer 0 to show absence of the number and integer 1 to show the presence of the number in the array. The function should have three parameters: (1) an array AR (2) the number to be searched and (3) the number of elements N in the array.**

Ans.

```
int Lsearch(int AR[10],int DATA,int N)
{     for(int i=0;i<n;i++)
      {     if(AR[i]==DATA)  return i; //return index of item in case of
successful search
      }
      return -1; //the control will reach here only when item is not found
}
```

12. **Suppose a one dimensional array ARR containing integers is arranged in ascending order. Write a user defined function in C++ to search for one integer from ARR with the help of binary search method, returning an integer 0 to show absence of the number and integer 1 to show the presence of the number in the array. The function should have three parameters: (1) an array ARR (2) the number DATA to be searched and (3) the number of elements N.**

Ans.

```
int bsearch(int ARR[10],int DATA,int N)
{     int beg=0,last=N-1,mid;
      while(beg<=last)
      {     mid=(beg+last)/2;
            if(ARR[mid]==DATA)  return 1;     //element is present in array
                  else if(DATA>ARR[mid]) beg=mid+1;
            else last=mid-1;
      }
      return 0;  //element is absent in array
}
```

13. **Suppose A, B, C are arrays of integers of size M, N and M+N respectively. The numbers in array A appear in ascending order while the numbers in array B appear in descending order. Write a user defined function in C++ to produce third array C by merging arrays A ad B in Ascending order. Use A, B, and C as arguments in the function.**

Ans.

```
#include<iostream.h>
void Merge(int A[],int M,int B[],int N,int C[]);
```

```
int mai()
{   int A[50],B[50],C[50],MN=0,M,N;
    cout<<"How many elements do U want to create first array with? ";
    cin>>M;
    cout<<"Enter First Array's elements [ascending]...";
    for(int i=0;i<M;i++)
        cin>>A[i];
    cout<<"How many elements do U want to create second array with? ";
    cin>>N;
    MN=M+N;
    cout<<"Enter Second Array's elements [descending]...";
    for(int i=0;i<N;i++)
        cin>>B[i];
    Merge(A,M,B,N,C);
    cout<<"The merged array is....";
    for(i=0;i<MN;i++)
        cout<<C[i]<<"  ";
    cout<<endl;
    return 0;
}
void Merge(int A[],int M,int B[],int N,int C[])
{   int a,b,c;
    for(a=0,b=-1,c=0;a<M&&b>=0;)
    {
        if(A[a]<=B[b]) C[c++]=A[a++];
            else C[c++]=B[b--];
    }
    if(a<M)
    {   while(a<M)
            C[c++]=A[a++];
    }
    else
    {   while(b>=0)
            C[c++]=B[b--];
    }
}
```

14. **Suppose X, Y, Z are arrays of integers of size M, N and M+N respectively. The numbers in array X and Y appear in descending order. Write a user defined function in C++ to produce third array Z by merging arrays X and Y in descending order.**

Ans.

```
#include<iostream.h>
void Merge(int X[],int M,int Y[],int N,int Z[]);
int main()
{   int X[50],Y[50],Z[50],MN=0,M,N;
    cout<<"How many elements do U want to create first array with? ";
    cin>>M;
    cout<<"Enter First Array's elements [descending]...";
    for(int i=0;i<M;i++)
        cin>>X[i];
    cout<<"How many elements do U want to create second array with? ";
    cin>>N;
    MN=M+N;
    cout<<"Enter Second Array's elements [descending]...";
    for(int i=0;i<N;i++)
        cin>>Y[i];
    Merge(X,M,Y,N,Z);
```

```
     cout<<"The merged array is....";
     for(i=0;i<MN;i++)
          cout<<Y[i]<<"  ";
     cout<<endl;
     return 0;
}
void Merge(int X[],int M,int Y[],int N,int Z[])
{    int x,y,z;
     for(x=-1,y=-1,z=-1;x>=0&&y>=0;)
     {
          if(X[x]<=Y[y]) Z[z--]=X[x--];
               else Z[z--]=Y[y--];
     }
     if(x<0)
     {    while(x>=0)
               Z[z--]=X[x--];
     }
     else
     {    while(y>=0)
               Z[z--]=Y[y--];
     }
}
```

15. **Given two arrays of integers X and Y of sizes m and n respectively. Write a function named MERGE() which will produce a third array named Z, such that the following sequence is followed:**
(i) All odd numbers of X from left to right are copied into Z from left to right
(ii) All even numbers of X from left to right are copied into Z from right to left
(iii) All odd numbers of Y from left to right are copied into Z from left to right
(iv) All even numbers of Y from left to right are copied into Z from right to left
X, Y and Z are passed as argument to MERGE().
e.g., X is {3, 2, 1, 7, 6, 3} and Y is {9, 3, 5, 6, 2, 8, 10}
the resultant array Z is {3, 1, 7, 3, 9, 3, 5, 10, 8, 2, 6, 6, 2}

Ans.

```
void MERGE(int X[],int Y[],int n,int m)
{    int Z[20],i=0,j=0,k=0,l=m+n-1;
     while(i<n&&k<20)
     {    if(X[i]%2!=0)
          {    Z[k]=X[i];
               k++;
               i++;
          }
          else
          {    Z[l]=X[i];
               l--;
               i++;
          }
     }
     while(j<m&&k<20)
     {    if(Y[j]%2!=0)
          {    Z[k]=Y[j];
               k++;
               j++;
          }
          else
          {    Z[l]=Y[j];
               l--;
```

```
                j++;
            }
        }
        cout<<"The elements of an array C is:";
        for(i=0;i<n+m;i++)
            cout<<"\n"<<Z[i];
}
```

16. **Assume an array E containing elements of structure Employee is required to be arranged in descending order of Salary. Write a C++ function to arrange the same with the help of bubble sort, the array and its size is required to be passed as parameters to the function. Definition of structure Employee is as follows:**

```
        struct Employee
        {
            int Eno;
            char Name[25];
            float Salary;
        };
```

Ans.

```
void Sort_Sal (Employee E[ ], int N)
{
     Employee Temp;
     for (int I=0; I<N-1;I++)
        for (int J=0;J<N-I-1;J++)
            if (E[J].Salary <E[J+1]. Salary)
            {
                 Temp = E[J];
                 E[J] = E[J+1];
                 E[J+1] = Temp;
            }
}
```

17. **Write a DSUM() function in C++ to find sum of Diagonal Elements from N x M Matrix. (Assuming that the N is a odd numbers)**

Ans.

```
int DSUM(int A[],int N)
{       int i,dsum1=0,dsum2=0;
        for(i=0;i<N;i++)
        {       dsum1+=A[i][i];
                dsum2+=A[N-(i+1)][i];
        }
        return(dsum1+dsum2-A[N/2][N/2]);
                //because middle element is added twice
}
```

18. **Given two arrays of integers A and B of sizes M and N respectively. Write a function named MIX() which will produce a third array named C, such that the following sequence is followed:**

(i) All even numbers of A from left to right are copied into C from left to right

(ii) All odd numbers of A from left to right are copied into C from right to left

(iii) All even numbers of B from left to right are copied into C from left to right

(iv) All odd numbers of B from left to right are copied into C from right to left

X, Y and Z are passed as argument to MERGE().

e.g., A is {3, 2, 1, 7, 6, 3} and B is {9, 3, 5, 6, 2, 8, 10}

the resultant array C is {2, 6, 6, 2, 8, 10, 5, 3, 9, 3, 7, 1, 3}

Ans.

```
void MIX(int A[],int B[],int n,int m)
{       int C[20],i=0,j=0,k=0,l;
        l=m+n-1;
        while(i<n&&k<20)
        {       if(A[i]%2==0)
                {       C[k]=A[i];
```

```
                  k++;
                  i++;
            }
            else
            {     C[l]=A[i];
                  l--;
                  i++;
            }
      }
      while(j<m&&k<20)
      {     if(B[j]%2==0)
            {     C[k]=B[j];
                  k++;
                  j++;
            }
            else
            {     C[l]=B[j];
                  l--;
                  j++;
            }
      }
      cout<<"The elements of an array C is:";
      for(i=0;i<n+m;i++)
            cout<<"\n"<<C[i];
}
```

19. **Suppose an array P containing float is arranged in ascending order. Write a user defined function in C++ to search for one float from P with the help of binary search method. The function should return an integer 0 to show absence of the number and integer 1 to show the presence of the number in the array. The function should have three parameters: (1) an array P (2) the number DATA to be searched and (3) the number of elements N.**

Ans.

```
int bsearch(float P[10],int DATA,int N)
{     int beg=0,last=N-1,mid;
      while(beg<=last)
      {     mid=(beg+last)/2;
            if(P[mid]==DATA)  return 1;     //element is present in array
                  else if(DATA>P[mid]) beg=mid+1;
            else last=mid-1;
      }
      return 0;  //element is absent in array
}
```

20. **Write a function in C++, which accepts an integer array and its size as arguments and swap the elements of every even location with its following odd location.**

Example: if an array of nine elements initially contains the elements as 2, 4, 1, 6, 5, 7, 9, 23, 10

then the function should rearrange the array as 4, 2, 6, 1, 7, 5, 23, 9, 10

Ans.

```
void ElementSwap(int A[],int size)
{     int lim,tmp;
      if(size%2!=0)    //if array has odd no. of element
            lim=size-1;
      else
            lim=size;
      for(int i=0;i<lim;i+=2)
      {     tmp=A[i];
            A[i]=A[i+1];
            A[i+1]=tmp;
      }
}
```

21. **Write a function in C++, which accepts an integer array and its size as arguments and replaces elements having odd values with thrice its value and elements having even values with twice its value.**
Example: if an array of nine elements initially contains the elements as 3, 4, 5, 16, 9
then the function should rearrange the array as 9, 8, 15, 32, 27

Ans.

```
void RearrangeArray(int A[],int size)
{
      for(int i=0;i<size;i++)
      {     if(A[i]%2==0)
                  A[i]*=2;
            else
                  A[i]*=3;
      }
}
```

22. **Write a function in C++ to print the product of each column of a two dimensional integer array passed as the argument of the function.**
Explain: if the two dimensional array contains

1	**2**	**4**
3	**5**	**6**
4	**3**	**2**
2	**1**	**5**

Then the output should appear as:

Product of Column 1 = 24
Product of Column 2 = 30
Product of Column 3 = 240

Ans.

```
void ColProd(int A[4][3],int r,int c)
{     int Prod[C],i,j;
      for(j=0;j<c;j++)
      {     Prod[j]=1;
            for(i=0;i<r;i++)
                  Prod[j]*=A[i][j];
            cout<<"Product of Column" <<j+1<<"="<<Prod[j]<<endl;
      }
}
```

23. **Write a function in C++ which accepts a 2D array of integers and its size as arguments and display the elements which lie on diagonals.**
[Assuming the 2D Array to be a square matrix with odd dimension i.e., 3 x 3, 5 x 5, 7 x 7 etc….]
Example, if the array content is

5 4 3
6 7 8
1 2 9

Output through the function should be:

Diagonal One: 5 7 9
Diagonal Two: 3 7 1

Ans.

```
const int n=5;
void Diagonals(int A[n][n], int size)
{
      int i,j;
      cout<<"Diagonal One:";
      for(i=0;i<n;i++)
            cout<<A[i]ij]<<" ";
      cout<<"\n Diagonal Two:"
```

```
     for(i=0;i<n;i++)
          cout<<A[i][n-(i+1)]<<" ";
}
```

24. **Write a function in C++ which accepts a 2D array of integers and its size as arguments and display the elements of middle row and the elements of middle column.**
[Assuming the 2D Array to be a square matrix with odd dimension i.e., 3 x 3, 5 x 5, 7 x 7 etc….]
Example, if the array content is

```
3 5 4
7 6 9
2 1 8
```

Output through the function should be:
Middle Row: 7 6 9
Middle Column: 5 6 1

Ans.

```
const int S=7;   // or it may be 3 or 5
int DispMRowMCol(int Arr[S][S],int S)
{    int mid=S/2;
     int i;
     //Extracting middle row
     cout<<"\n Middle Row:";
     for(i=0;i<S;i++)
          cout<<Arr[mid][i]<<" ";
     //Extracting middle column
     cout<<"\n Middle Column:";
     for(i=0;i<S;i++)
          cout<<Arr[i][mid]<<" ";
}
```

25. **Write a function in C++ which accepts a 2D array of integers and its size as arguments and swaps the elements of every even location with its following odd location.**
Example: if an array of nine elements initially contains the elements as 2, 4, 1, 6, 5, 7, 9, 23, 10
then the function should rearrange the array as
4, 2, 6, 1, 7, 5, 23, 9, 10

Ans. Same as Q-20 of Long Answer Question.

26. **Write a function in C++ to print the product of each row of a two dimensional integer array passed as the argument of the function.**
Explain: if the two dimensional array contains

20	40	10
40	50	30
60	30	20
40	20	30

Then the output should appear as:
Product of Diagonal 1 = (1 x 5 x 2 x 4)=40
Product of Diagonal 2 = (3 x 6 x 3 x 2)=108

Ans.

```
void RowProduct(int A[4][3],int R,int C)
{    int Prod[R];
     for(int i=0;i<R;i++)
     {    Prod[i]=1;
          for(int j=0;j<c;j++)
               Prod[i]*=A[i][j];
          cout<<"Product of row"<<i+1<<"="<<Prod[i]<<endl;
     }
}
```

27. **Write a function REASSIGN() in C++, which accepts an array of integer and its size as parameters and divide all**

those array elements by 5 which are divisible by 5 and multiply other array element by 2.

Sample Input Data of the array

A[0]	A[1]	A[2]	A[3]	A[4]
20	12	15	60	32

Content of the array after calling REASSIGN() function

A[0]	A[1]	A[2]	A[3]	A[4]
4	24	3	12	64

Ans.

```
void REASSIGN (int Arr[ ], int Size)
 {
      for (int i=0;i<Size;i++)
          if (Arr[i]%5==0)
                Arr[i]/=5;
          else
                Arr[i]*=2;
}
```

28. **Write a function SORTSCORE() in C++ to sort an array of structure Examinee in descending order of Score using Bubble Sort.**

Note. Assume the following definition of structure Examinee

```
struct Examinee
{     long RollNo;
      char Name[20];
      float Score;
};
```

Sample Content of the array (before sorting)

RollNo	Name	Score
1001	Ravyank Kapur	300
1005	Farida Khan	289
1002	Anika Jain	345
1003	George Peter	297

Sample Content of the array (after sorting)

RollNo	Name	Score
1002	Anika Jain	345
1001	Ravyank Kapur	300
1003	George Peter	297
1005	Farida Khan	289

Ans.

```
void SORTSCORE(Examinee E[ ], int N)
{
     Examinee Temp;
     for (int I=0; I<N-1;I++)
        for (int J=0;J<N-I-1;J++)
            if (E[J].Score <E[J+1]. Score)
            {
                 Temp = E[J];
                 E[J] = E[J+1];
                 E[J+1] = Temp;
            }
}
```

29. **Write a function SORTPOINTS() in C++ to sort an array of structure Game in descending order of Points using Bubble Sort.**

Note. Assume the following definition of structure Game

```
struct Game
{     long PNo;    //Player Number
```

```
            char PName[20];
            float Points;
       };
```

Sample Content of the array (before sorting)

PNo	PName	Points
103	Ritika Kapur	3001
104	John Philip	2819
101	Razia Abbas	3451
105	Tarun Kumar	2971

Sample Content of the array (after sorting)

RollNo	Name	Score
101	Razia Abbas	3451
103	Ritika Kapur	3001
105	Tarun Kumar	2971
104	John Philip	2819

Ans.

```
void SORTPOINTS(Game G[ ], int N)
{
     Game Temp;
     for (int I=0; I<N-1;I++)
        for (int J=0;J<N-I-1;J++)
            if (G[J].Points <G[J+1].Points)
            {
                 Temp = G[J];
                 G[J] = G[J+1];
                 G[J+1] = Temp;
            }
}
```

30. **Define a function SWAPCOL() in C++ to swap (interchange) the first column elements with the last column elements, for a two dimensional integer array passed as the argument of the function.**

Example: If the two dimensional array contents

2	1	4	9
1	3	7	7
5	8	6	3
7	2	1	2

After swapping of the content of 1st column and last column, it should be:

9	1	4	2
7	3	7	1
3	8	6	5
2	2	1	7

Ans.

```
void SWAPCOL(int A[ ][100], int M, int N)
{
   int Temp, I;
   for (I=0;I<M;I++)
   {
        Temp = A[I][0];
        A[I][0] = A[I][N-1];
        A[I][N-1] = Temp;
   }
}
```

31. **Define a function SWAPARR() in C++ to swap (interchange) the first row elements with the last row elements, for a two dimensional integer array passed as the argument of the function.**

Example: If the two dimensional array contents

5	6	3	2
1	2	4	9
2	5	8	1
9	7	5	8

After swapping of the content of 1st column and last column, it should be:

9	7	5	8
1	2	4	9
2	5	8	1
5	6	3	2

Ans.

```
void SWAPARR (int A[100][], int M, int N)
{
   int Temp, I;
   for (I=0;I<M;I++)
   {
        Temp = A[0][I];
        A[0][I] = A[N-1][I];
        A[N-1][I] = Temp;
   }
}
```