[DATA STRUCTURES]
Chapter - 02 : "Arrays"

▪ ARRAYS

"An array is a collection of variables of the same type that are referenced by a common name."

Arrays are a way to group a number of items into a larger unit. In C, array elements are stored in contiguous (consecutive) memory locations. The lowest address corresponds to the first element and the highest address to the last element. Arrays can have data items of simple types like **int** or **float** or even of user-defined types like structure or objects.

Types of Arrays

Arrays are of different types:

1. **Single-dimensional arrays,** comprised of finite homogenous(same type) elements.
2. **Multi-dimensional arrays,** comprised of elements, each of which is itself an array. A two dimensional array is the simplest of the multi-dimensional arrays. However, C programming allows arrays of more than two dimensions. The exact limit (of dimensions), if any, is determined by the compiler you use.

SINGLE DIMENSIONAL ARRAYS

The simplest form of an array is the **single-dimensional array**. The array is given a name and its elements are referred to by their subscripts or indices. C language array's index numbering starts with zero. The index of first element is known as **lower bound** and the index of the last element is known as **upper bound**.

Declaration of Single-dimensional array :

data_type array-name[size];

where *data_type* declares the base type of array, which is the type of each element in the array. The *array-name* specifies the name with which the array will be referenced and *size* defines how many elements the array will hold. The size must be an integer value or integer constant without any sign.

For e.g.

```
int marks[10];
```

The above statement declared array **marks** with 10 elements, marks[0] to marks[9].

Initialization of array :

data_type array-name[size]={element-1,element-2,……..,element-n};

or

data_type array-name[ ]={element-1,element-2,……..,element-n};

For example,

```
int marks[5]={50,25,72,45,30};
```

```
Marks[0]=50;
Marks[1]=25;
Marks[2]=72;
Marks[3]=45;
Marks[4]=30;
```

```
or int marks[ ]={50,25,72,45,30};
Also float price[ ] = {300.50, 250.50, 500.50, 175.50, 900.50};
and  char grade[ ] = {'D' , 'A' , 'C' , 'B' , 'A' , 'C' };
```

Accessing an element at a particular index for one dimensional arrays

Individual element of an array can be accessed using the following syntax :

array_name[index or subscript];

For example, to assign a value to second location of array, we give the following statement

```
marks[1]=90;
```

Similarly, for reading the value of fourth element in array_name **marks**, we give the following statement :

```
scanf("%d",&marks[3]);
```

For writing the value of second element in array_name **marks**, we give the following statement :

```
printf("%d\t",marks[1]);
```

Arrays can always be read or written through loop. If we read a one-dimensional array, it requires one loop for reading and other for writing (printing) the array. For example :

(a) For reading the array

For reading the marks of 10 students :

```
for ( i = 0; i < = 9 ; i++)
{
   scanf("%d", & marks [ i ] );
}
```

(b) For writing the array

```
for ( i = 0; i < =9 ; i++)
{
   printf("%d\t", marks [ i ] );
}
```

Implementation of one-dimensional array in memory :

The address of a particular element in one-dimensional array is given by the relation :

*Address of element a[k] = B+ W * k*

where **B** is the base address of the array, **W** is the size of each element in array, and **k** is the number of required element in the array (index of element) which should be a integer quantity. For example :

Let the base address of first element of the array is 2000 (i.e. base address B is = 2000), and each element of the array occupies four bytes in the memory, then address of fifth element of a one-dimensional array a[4] will be given as :

```
Address of element a[4] = 2000 + 4 * 4
                        = 2000 + 16
                        = 2016
```

PROGRAMS (SINGLE DIMENSIONAL ARRAYS)

Program 1 : WAP for array initialization

```
#include<stdio.h>
#include<conio.h>
void main( )
{
clrscr( );
// Array declaration and initialization
int marks[ ]={50,60,70,80,90},i;

// Array output
printf("\nMarks of 5 students are : \n");
for(i=0;i<5;i++)
 printf("%d\t",marks[i]);
getch( );
}
```

OUTPUT :

```
Marks of 5 students are :
50      60      70      80      90
```

Program 2 : WAP for array input from user

```
#include<stdio.h>
#include<conio.h>
void main( )
{
clrscr( );
// Array declaration
int marks[5],i;

// Array input
printf("Enter marks of 5 students : \n");
for(i=0;i<5;i++)
 scanf("%d",&marks[i]);

// Array output
printf("\nMarks of 5 students are : \n");
for(i=0;i<5;i++)
 printf("%d\t",marks[i]);
getch( );
}
```

OUTPUT :

```
Enter marks of 5 students :
50      60      70      80      90

Marks of 5 students are :
50      60      70      80      90
```

Program 3 : WAP to display the sum and average of elements of an array

```
#include<stdio.h>
#include<conio.h>
void main( )
{
clrscr( );
int A[5],i;
float sum=0,avg;
printf("Enter 5 elements of an array : \n");
for(i=0;i<5;i++)
  scanf("%d",&A[i]);
for(i=0;i<5;i++)
  sum=sum+A[i];
avg=sum/5;
printf("\nSum : %.2f",sum);
printf("\nAverage : %.2f",avg);
getch( );
}
```

OUTPUT :

```
Enter 5 elements of an array :
2       4       6       8       10

Sum : 30.00
Average : 6.00
```

Program 4 : WAP to display the largest and smallest element of an array

```
#include<stdio.h>
#include<conio.h>
void main( )
{
clrscr( );
```

```
int A[5],i,max,min;
printf("Enter 5 elements of an array : \n");
for(i=0;i<5;i++)
  scanf("%d",&A[i]);
max=min=A[0];
for(i=0;i<5;i++)
{
 if(max<A[i])
  max=A[i];
 if(min>A[i])
  min=A[i];
}
printf("\nLargest element in array : %d",max);
printf("\nSmallest element in array : %d",min);
getch( );
}
```

OUTPUT :

```
Enter 5 elements of an array :
3       9       5       1       8

Largest element in array : 9
Smallest element in array : 1
```

Program 5 : WAP to insert a number in an array

```
#include<stdio.h>
#include<conio.h>
void main( )
{
clrscr( );
int len;
printf("Enter size of array (max. 10) : ");
scanf("%d",&len);

int A[10],num,i,pos;
printf("\nEnter %d elements of array : \n",len);
for(i=0;i<len;i++)
  scanf("%d",&A[i]);

printf("\nOriginal array is : \n");
for(i=0;i<len;i++)
  printf("%d\t",A[i]);
```

```
printf("\n\nEnter the element to be inserted : ");
scanf("%d",&num);
printf("Enter the position of insertion : ");
scanf("%d",&pos);
pos--;

for(i=len-1;i>=pos;i--)  // Shifts down one position
 A[i+1]=A[i];

A[pos]=num;

if(pos>len)
 printf("\nInsertion outside the array");
else
{
 printf("\nNew array after insertion : \n");
 len++;
 for(i=0;i<len;i++)
   printf("%d\t",A[i]);
}
getch( );
}
```

OUTPUT :

```
Enter size of array (max. 10) : 5

Enter 5 elements of array :
2       4       8       10      12

Original array is :
2       4       8       10      12

Enter the element to be inserted : 6
Enter the position of insertion : 3

New array after insertion :
2       4       6       8       10      12
```

Program 6 : WAP to delete a number from an array

```
#include<stdio.h>
#include<conio.h>
void main( )
{
clrscr( );
int A[10],i,len,num,f=0;
printf("Enter the size of array (max. 10) : ");
scanf("%d",&len);
printf("\nEnter %d elements of an array : \n",len);
for(i=0;i<len;i++)
 scanf("%d",&A[i]);
printf("\nOriginal array is : \n");
for(i=0;i<len;i++)
 printf("%d\t",A[i]);
printf("\n\nEnter the element to delete : ");
scanf("%d",&num);
for(i=0;i<len;i++)
{
 if(num==A[i])
  {
   f=1;
   for(;i<len-1;i++)
     A[i]=A[i+1];
   len--;
   break;
  }
}
if(f==0)
 printf("\nNumber not found in array");
else
{
printf("\nNew Array after deletion : \n");
for(i=0;i<len;i++)
 printf("%d\t",A[i]);
}
getch( );
}
```

OUTPUT :

```
Enter the size of array (max. 10) : 5

Enter 5 elements of an array :
2       4       6       8       10
```

Original array is :
2 4 6 8 10

Enter the element to delete : 6

New Array after deletion :
2 4 8 10

DOUBLE-DIMENSIONAL ARRAYS

A **double dimensional array** is an array in which each element is itself an array. For example, an array A[R][C] is an R by C table with R rows and C columns containing
R * C elements.

The number of elements in a two-dimensional array can be determined by multiplying number of rows with number of columns. For example, the number of element in an array A[4][3] is calculated as 4 * 3 = 12.

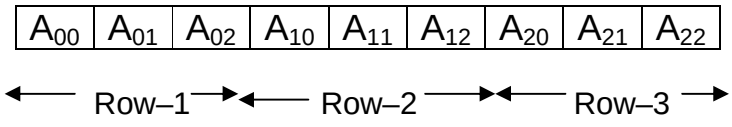
Implementation of Two-dimensional array in Memory

While storing the elements of a 2-D array in memory, these are allocated contiguous memory locations. A two-dimensional array can be implemented in a programming language in two ways :

1. Row-major implementation
2. Column-major implementation

Row-major implementation :

Row-major implementation is a linearization technique in which elements of array are readed from the keyboard row-wise i.e. the complete first row is stored, then the complete second row is stored and so on. For example, an array **A [3] [3]** is stored in the memory as shown in Fig.(1) below :



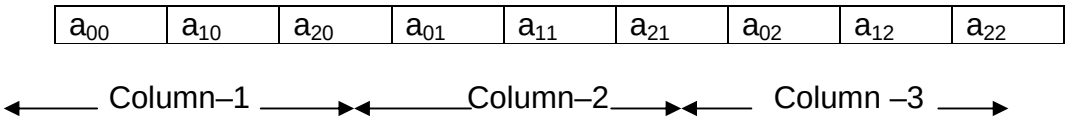
The storage can be clearly understood by arranging array as matrix as shown below :

$$\mathbf{a} = \begin{vmatrix} a_{00} & a_{01} & a_{02} \\ a_{10} & a_{11} & a_{12} \\ a_{20} & a_{21} & a_{22} \end{vmatrix} \quad \begin{matrix} \textbf{Row 1} \\ \textbf{Row 2} \\ \textbf{Row 3} \end{matrix}$$

Column-major implementation :

Column-major implementation is a linearization technique in which elements of array are reader from the keyboard column-wise i.e. the complete first column is stored, then the complete

second column is stored and so on. For example, an array a[3][3] is stored in the memory as shown in Fig.(1) below :



The storage can be clearly understood by arranging array as matrix as shown below :

$$\mathbf{a} = \begin{vmatrix} a_{00} & a_{01} & a_{02} \\ a_{10} & a_{11} & a_{12} \\ a_{20} & a_{21} & a_{22} \end{vmatrix}$$

Column 1 Column 2 Column 3

Double Dimensional Array declartion :

datatype arrayvariablename[rowsize][col.size];

For e.g.
int A[4][3];

where **int** is datatype, **A** is array variable_name, **4** is row size and **3** is columnsize.

Double Dimensional Array initialization :

datatype arrayname[rowsize][col. size]={{1st row elements},{2nd row elements},……};

For e.g.

int A[4][3]={{1,2,3}, {4,5,6,}, {7,8,9}, {10,11,12}};

Array input from user :

Row major form

```
for(r=0;r<4;r++)
 for(c=0;c<3;c++)
  scanf("%d",&A[r][c]);
```

Column major form

```
for(c=0;c<3;c++)
 for(r=0;r<4;r++)
  scanf("%d",&A[r][c]);
```

Array output :

```
for(r=0;r<4;r++)
{
 for(c=0;c<3;c++)
    printf("%d\t",A[r][c]);
  printf("\n");
}
```

Program 1 : Double Dimensional array (Matrix) Initialization & Output

```
#include<stdio.h>
#include<conio.h>
void main( )
{
clrscr( );
// Double Dim. Array declaration and initialization
int A[4][3]={{1,2,3},{4,5,6},{7,8,9},{10,11,12}};
int r,c;
// Double Dim. Array Output
printf("\nGiven 4 * 3 matrix is : \n");
for(r=0;r<4;r++)
{
 for(c=0;c<3;c++)
   printf("%d\t",A[r][c]);
printf("\n");
}
getch( );
}
```

OUTPUT :

```
Given 4 * 3 matrix is :
1       2       3
4       5       6
7       8       9
10      11      12
```

Program 2 : Double Dimensional array(Matrix) input & Output

```
#include<stdio.h>
#include<conio.h>
void main( )
{
clrscr( );
// Matrix declaration
int A[4][3],r,c;

// Matrix input
printf("\nEnter elements of a  4 * 3 matrix : \n");
for(r=0;r<4;r++)
 for(c=0;c<3;c++)
  scanf("%d",&A[r][c]);
// Matrix output
printf("\nGiven 4 * 3 matrix is : \n");
for(r=0;r<4;r++)
{
 for(c=0;c<3;c++)
   printf("%d\t",A[r][c]);
printf("\n");
}
getch( );
}
```

OUTPUT :

```
Enter elements of a  4 * 3 matrix :
1       2       3
4       5       6
7       8       9
10      11      12

Given 4 * 3 matrix is :
1       2       3
4       5       6
7       8       9
10      11      12
```

Program 3 : Addition of Two 3 * 3 Matrices

```
#include<stdio.h>
#include<conio.h>
void main( )
{
clrscr( );
// Matrix declaration
int A[3][3],B[3][3],C[3][3],r,c;

// Matrix input
printf("\nEnter elements of first 3 * 3 matrix : \n");
for(r=0;r<3;r++)
 for(c=0;c<3;c++)
  scanf("%d",&A[r][c]);
printf("\nEnter elements of second 3 * 3 matrix : \n");
for(r=0;r<3;r++)
 for(c=0;c<3;c++)
  scanf("%d",&B[r][c]);
// Matrix addition
printf("\nAddition of first two matrices : \n");
for(r=0;r<3;r++)
{
 for(c=0;c<3;c++)
 {
  C[r][c]=A[r][c]+B[r][c];
  printf("%d\t",C[r][c]);
 }
printf("\n");
}
getch( );
}
```

OUTPUT :

```
Enter elements of first  3 * 3 matrix :
1       2       3
4       5       6
7       8       9

Enter elements of second  3 * 3 matrix :
2       3       4
5       6       7
```

```
8       9       10

Addition of first two matrices :
3       5       7
9       11      13
15      17      19
```

Program 4 : WAP to display the transpose of a 3 * 3 matrix

```
#include<stdio.h>
#include<conio.h>
void main( )
{
clrscr( );
int A[3][3],r,c;
printf("\nEnter elements of a 3 * 3 matrix :\n");
for(r=0;r<3;r++)
  for(c=0;c<3;c++)
   scanf("%d",&A[r][c]);
printf("\nOriginal matrix is :\n");
for(r=0;r<3;r++)
{
  for(c=0;c<3;c++)
   printf("%d\t",A[r][c]);
printf("\n");
}

printf("\nTranspose of given matrix is :\n");
for(r=0;r<3;r++)
{
  for(c=0;c<3;c++)
   printf("%d\t",A[c][r]);
printf("\n");
}
getch( );
}
```

OUTPUT :

```
Enter elements of a 3 * 3 matrix :
1       2       3
4       5       6
7       8       9
```

Original matrix is :
1 2 3
4 5 6
7 8 9

Transpose of given matrix is :
1 4 7
2 5 8
3 6 9

Program 5 : WAP to multiply two 3 * 3 matrices

```
#include<stdio.h>
#include<conio.h>
void main( )
{
clrscr( );
int A[3][3],B[3][3],C[3][3],r,c,k;
printf("Enter elements of first 3 * 3 matrix : \n");
for(r=0;r<3;r++)
 for(c=0;c<3;c++)
  scanf("%d",&A[r][c]);

printf("\nEnter elements of second 3 * 3 matrix : \n");
for(r=0;r<3;r++)
 for(c=0;c<3;c++)
  scanf("%d",&B[r][c]);
printf("\nProduct of first two 3 * 3 matrices :\n");
for(r=0;r<3;r++)
{
 for(c=0;c<3;c++)
 {
  C[r][c]=0;
   for(k=0;k<3;k++)
    C[r][c]=C[r][c]+(A[r][k]*B[k][c]);
  printf("%d\t",C[r][c]);
 }
printf("\n");
}
getch( );
}
```

OUTPUT :

```
Enter elements of first 3 * 3 matrix :
1       2       3
4       5       6
7       8       9

Enter elements of second 3 * 3 matrix :
1       2       3
4       5       6
7       8       9

Product of first two 3 * 3 matrices :
30      36      42
66      81      96
102     126     150
```

Multidimensional arrays :

The general syntax of a multidimensional array is :

datatype arrayname[size-1][size-2]……[size-n];

For example :

```
int A[5][2][3];
float B[2][5][3];
```

The simplest form of a multidimensional array is a two-dimensional array, which is also known as array of an array.

Sparse Matrices :

If many of elements from a m x n matrix have a value 0 then the matrix is known as **sparse matrix.** A matrix that is not sparse is known as a **dense matrix**. There is no precise definition of when a matrix is sparse and when it is not, but it is a concept, which we can all recognize naturally. If the matrix is sparse, we must consider an alternative way of representing it rather than a normal row major or column major arrangement. This is because if majority of elements of the matrix are 0 then the alternative through which we can store only the non-zero elements and keep intact the functionality of the matrix can save a lot of memory space. Fig. (3) shows sparse matrix 5 * 6 with 5 non zero elements.

0	0	0	6	0	0
0	0	3	0	0	0
0	8	0	0	2	0
0	0	0	0	0	0
0	0	0	0	9	0

Assignment of data structure

Name: Muhammad Irfan

Roll: 221108

Section: A

Dept: Software Engineering

Topic : 1: Implementation of quick sort and its algorithm

2: implementation of tower of Hanoi and its algorithm

1: quick sort program implementation:

```
#include <iostream>
using namespace std;
int partition (int a[], int start, int end)
{
   int pivot = a[end]; // pivot element
   int i = (start - 1);

   for (int j = start; j <= end - 1; j++)
```

```
    {   if (a[j] < pivot)
        {    i++;
            int t = a[i];
            a[i] = a[j];
            a[j] = t;
        }
    }
    int t = a[i+1];
    a[i+1] = a[end];
    a[end] = t;
    return (i + 1);
}

void quick(int a[], int start, int end)
{
    if (start < end)
    {
        int p = partition(a, start, end);
        quick(a, start, p - 1);
        quick(a, p + 1, end);
    }
}
```

```
void printArr(int a[], int n)
{
   int i;
   for (i = 0; i < n; i++)
      cout<<a[i]<< " ";
}
int main()
{
   int a[] = { 23, 8, 28, 13, 18, 26 };
   int n = sizeof(a) / sizeof(a[0]);

   cout<<"Before sorting array elements are - \n";
   printArr(a, n);
   quick(a, 0, n - 1);
   cout<<"\nAfter sorting array elements are - \n";
   printArr(a, n);

   return 0;
}
```

Tower of Hanoi code implementation:

```
include <iostream>
using namespace std;
```

```
void TOH(int n, char Sour, char Aux, char Des)
{
   if (n == 1) {
      cout << "Move Disk " << n << " from " << Sour << " to " << Des << endl;
      return;
   }

   TOH(n - 1, Sour, Des, Aux);
   cout << "Move Disk " << n << " from " << Sour << " to " << Des << endl;
   TOH(n - 1, Aux, Sour, Des);
}
int main()
{
   int n;

   cout << "Enter no. of disks:";
   cin >> n;
   TOH(n, 'A', 'B', 'C');

   return 0;
}
```

Algorithm of tower of Hanoi:

This is a non recursive solution of the tower of hanoi

TOWER(N,BEG,AUX,END)

1.Set TOP:=NULL

If N==1 then:

(A) Write BEG->END
(B) GO to step5
[End of if structure]

2. [Translate if Call tower (N-1,END BEG,AUX)"]

(a) [push current values and new return address onto stack .]

(i) Set TOP = TOP+1

(ii) Set STN[TOP]:=N,STBEG[TOP]:=END,

STAUX[TOP]:=AUX,STEND[TOP]:=END

(b)[Reset parameter]

Set N = N-1,BEG:=BEG ,AUX:=END,END:=AUX,

3. Write BEG->END

4.[Translation of call TOWER(N-1,AUX,BEG,END)"]

(a)[Push current values and new return address onto stack.]

(i)Set top = TOP+1;

(ii) Set STN[TOP]:=N,STBEG[TOP]:=BEG

STAAUX[AUX]:=AUX,STEND[TOP]:=END

STADD[TOP]:=5

(b) [Reset the parameter]

Set :=N= N-1,BEG:=AUX,AUX:=BEG,END:=END

(c) go to step 1

5. [Translate of return,”]

(a) IF TOP ==NULL then return.

(b) [Restore top values on stack]

(i) Set N :=STN[TOP] BEG:=STBEG[TOP]

ADD:=STADD[TOP]

(ii) Set TOP:= TOP-1

(c) Go to step Add

Algorithm of quick sort :

QUICK(A,N,BEG,END,LOC)

1.[Initialized] Set LEFT:=BEG.RIGHT:=END and LOC:=BEG

2.[Scan from right to left]

(a) Repeat while A[LOC]<=A[RIGHT] and LOC!=RIGHT:

RIGHT RIGHT -1

[End of loop]

(b) if LOC =RIGHT then Return

(c) if A[LOC]>A[RIGHT] then

(i) [interchange A[LOC] and A[RIGHT].]

TEMP:=A[LOC],and A[LOC]:=A[RIGHT]

(ii) A[RIGHT]:=TEMP

(iii) Go to step 3

3. [Scan from left to right]

(a) repeat while loop A[LEFT]<=A[LOC] and LEFT!=LOC:

LEFT = LEFT+1

[End of loop]

(b)If LOC = LEFT then Return

(c) If A[LEFT]>A[LOC] then

(i) [Interchange A[LEFT] and A[LOC].]

TEMP:=A[LOC],A[LOC]:=A[LEFT]

A[LEFT]:=TEMP

(ii) Set LOC:=TEMP

(iii) Go to step 2

[End od if structure]

BRAVE GROUP OF TYRES COMPANY

ADDRESS: Dir Chitral Road New Adda Wari (Opposite to Japan Tyres Center).

PHONE: 0307-9899940, 0315-9047924 .

EMAIL: nadeemkhan754323@gmail.com.

PROPRIETOR: Nadeem Khan and Co .

BUSINESS: Importer and Wholesaler of Tyres and Tubes.