UNIT – 6 Part I
Arrays

C Programming (CSC115)
BSc CSIT First Semester (TU)

Prepared by:
Dabbal Singh Mahara
ASCOL (2025)

ARRAYS IN C

- When we need to store multiple values of same type like *names of 10 students*, *50 mobile numbers*, *weight of 100 people*. In this case, to declare and manage different variables to store separate values are really tough and unmanageable.
- **Then, what?**
- C programming language provides an amazing feature to deal with such kind of situations that is known as **"Arrays"**.
- Array is a derived data type which is created with the help of basic data type.
- An array takes contiguous memory blocks to store series of values.

3 ARRAY IN C

- An array is a group of related data items that share a common name.
- The individual data items are called elements of the array and all of them are of same data type.
- The individual elements are characterized by array name followed by one or more subscripts (or indices) enclosed in square brackets

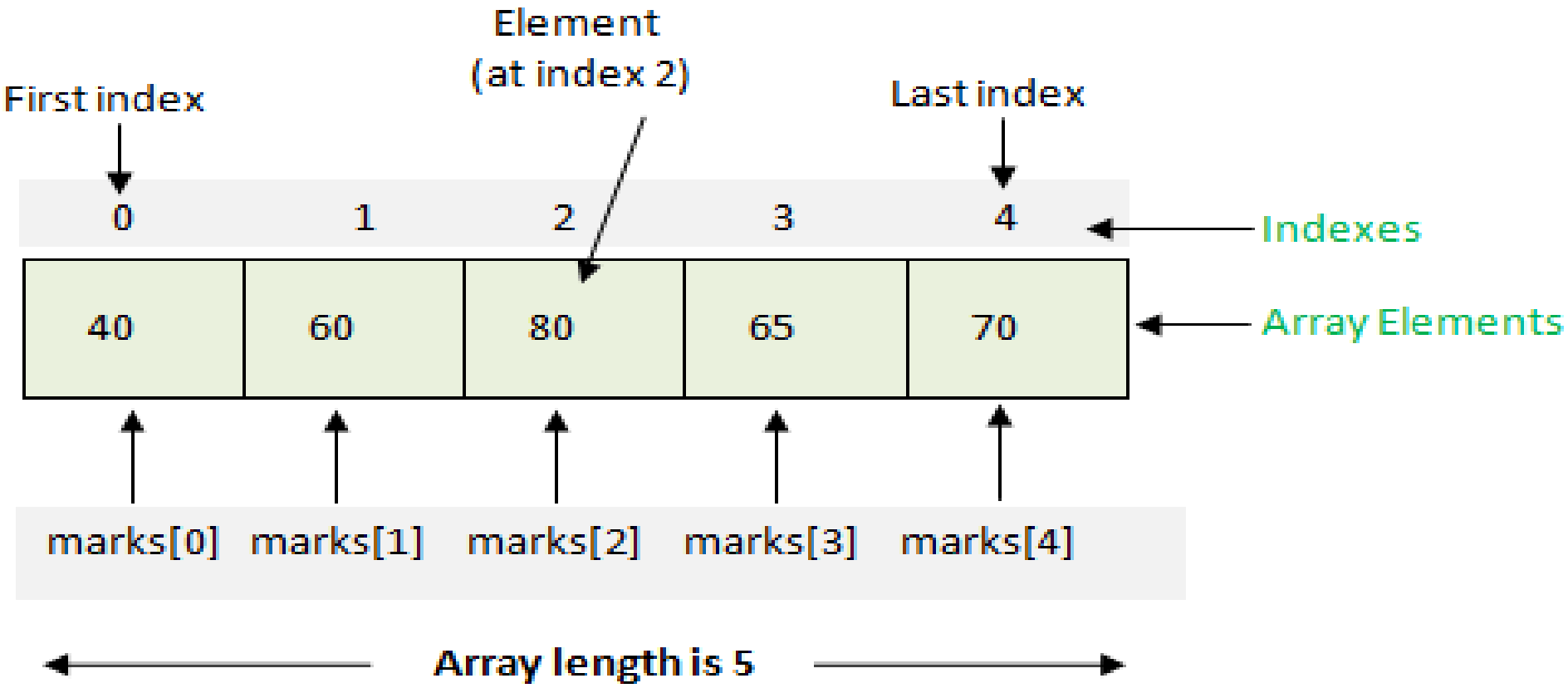
TYPES OF ARRAY

- There are two forms of the array in C.
  - One Dimensional Array
  - Multi Dimensional Array

ONE-DIMENSIONAL (1-D) ARRAY

- A list of items can be given one variable name using only one subscript (or dimension or index) and such a variable is called a *single-subscripted variable* or a *one-dimensional array*.
- The value of the single subscript or index from 0 to n-1 refers to the individual array elements; where n is the size of the array.
- E.g. the declaration *int a[4];* is a 1-D array of integer data type with 4 elements: *a[0]*, *a[1]*, *a[2]* and *a[3]*.

EXAMPLE: ARRAY



DECLARATION OF 1-D ARRAY

- Syntax:

  storage_class data_type array_name[size];

- *storage_class* refers to the *storage class* of the array. It may be auto, static, extern and register. It is optional. (Will be studied later!!)
- *data_type* is the *data type* of array. It may be *int*, *float*, *char*,…, etc. (Note: An *int* type array stores all data items of integer type.)
- *array_name* is the name of the array. Any valid name of a variable can be provided.
- *size* of the array is the number of elements in the array and is mentioned within square bracket. The size must be an integer constant like 100 or a symbolic constant (if symbolic constant SIZE is defined as #define SIZE 100, then array can be defined as int a[SIZE];)

DECLARATION OF 1-D ARRAY…

- Examples:
- *long marks[8];* i.e. marks is a long integer array of size 8. It can store 8 long values in *marks[0], marks[1],…, marks[7].*
- *float salary[50];* i.e. salary is a float array of size 50. It can store 50 fractional numbers in *salary[0], salary[1],…, salary[49].*
- *char name[10];* i.e. name is a character array of size 10. It can store 10 characters.

Note:"character array is also known as string"

INITIALIZATION OF 1-D ARRAY

- In an uninitialized array (declared only), the individual array elements contain garbage values.
- Assigning specific values to the individual array elements, at the time of array declaration, is known as *array initialization*.
- Since an array has multiple elements, braces are used to denote the entire array and commas are used to separate the individual values assigned to the individual elements in the array.

INITIALIZATION OF 1-D ARRAY…

- The syntax for array initialization is:

  storage_class data_type array_name[size]={value1, value2,…, valueN};

  where, *value1* is value of first element, *value2* is value of second element and so on.

- E.g.

  ```
  int a[5]={21, 13, 54, 5, 101};
  ```

  Here, a is an integer type array which has 5 elements. Their values are initialized to 21, 13, 54, 5 and 101 (i.e. a[0]=21, a[1]=13, a[2]=54, a[3]=5 and a[4]=101). These array elements are stored sequentially in separate memory locations.

INITIALIZATION OF 1-D ARRAY…

- Note:

```
int b[]={12, 75, 321};
```

Here, size of array b is not given, the compiler can automatically set its size according to the number of values given. Thus, the size of array b is 3 with its elements b[0], b[1] and b[2] initialized to values 12, 75, and 321 respectively. Therefore,

```
int b[]={12, 75, 321};  ≡  int b[3]={12, 75, 321};
```

- Note:

```
int c[10]={6 , 7, 12, 43, 0};
```

Here, size of array c is 10 but only 5 elements are assigned values at the time of initialization. In this case, all individual elements that are not assigned values contain zero as initial values. Thus, the value of c[5], c[6], c[7], c[8] and c[9] is zero.

ACCESSING ARRAY ELEMENTS

- A single operation involving an entire array, are not permitted in C.
- For e.g., if *num* and *list* are two similar arrays (i.e. same data type, dimension and size), then assignment operations, comparison operations, etc., involving these two arrays must be carried out on an element-by-element basis.

```
int a[5], b[5];
a=0;                    //RIGHT
b=a;                    //WRONG
if(a<b)                 //WRONG
{…}                     //WRONG
```

ACCESSING ARRAY ELEMENTS…

- However the following are RIGHT:

```
int a[5], b[5], x;
x=a[0]+10;
a[4]=a[1]+a[2];
b[3]=b[0]+a[3]+x;
b[4]=a[2]*6;
```

- The particular array element in an array is accessed by specifying the name of array, followed by a square bracket that encloses an integer, called array index.
- Generally, a loop is used to access (i.e. input and output) the elements of array.

//Program to access array elements

```
#include <stdio.h>
#include <conio.h>
main()
{
  int a[5],i;

   printf(“\nEnter 5 numbers:\t");
    for(i=0;i<5;i++)
    {
    scanf("%d", &a[i]);  //array input
    }
    printf("\nWe have entered these 5 numbers:\n");
    for(i=0;i<5;i++)
                {
                printf("\ta[%d]=%d", i, a[i]);    //array output
                }
  getch();
}
```

Note: C performs no bounds checking

Ensure array indices are within declared limits

```
//Program to show memory addresses of array elements
#include <stdio.h>
#include <conio.h>
int main()
{
float a[4]={20,0,5.6,98.5};
int i;

printf("The contiguous memory locations are:\t");
for(i=0;i<4;i++)
{
printf("\na[%d]=%f is located at\t%u.", i, a[i], &a[i]);
          //address of array elements
}
getch();
}
```

MULTIDIMENSIONAL ARRAYS

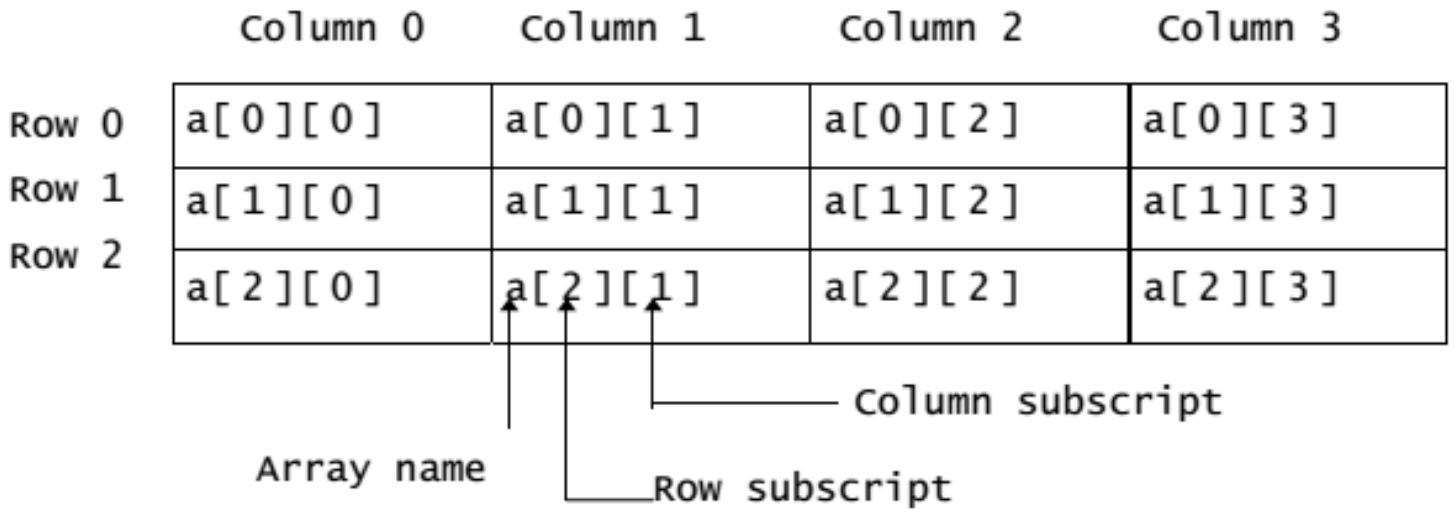
- Multidimensional arrays are those having more than one dimension.
- Multidimensional arrays are defined in the same way as one dimensional arrays, except that a separate pair of square brackets is required for each subscript or dimension or index.
- Thus a 2-D (two dimensional) array requires two pairs of square brackets, a 3-D array requires three pairs of square brackets and so on.

Two-Dimensional array

- Two-dimensional arrays are specified by two subscripts in the form of rows and columns
- Use first subscript to specify row no and the second subscript to specify column no.
- 2D-array is defined by following type declaration statement:

  datatype array_name[row_size][column_size];

- **int a[3][4]; this array can be thought as given below:**



MULTIDIMENSIONAL ARRAYS…

- Syntax for defining multidimensional array is:

storage_class data_type array_name[dim1][dim2]…[dimN];

Here, dim1, dim2,…,dimN are positive valued integer expressions that indicate the number of array elements associated with each subscript. Thus, total no. of elements=dim1*dim2*…*dimN

- E.g.

```
int survey[3][5][12];
```

Here, survey is a 3-D array that can contain 3*5*12=180 integer type data. This array survey may represent a survey data of rainfall during last three years (2009,2010,2011) from months Jan. to Dec. in five cities. Its individual elements are from survey[0][0][0] to survey[2][4][11].

2-D ARRAYS (INITIALIZATION)

- Like 1-D arrays, 2-D arrays can be initialized at the time of array definition or declaration.
- E.g.

```
int marks[2][3]={0,0,0,1,1,1};
```

Here, elements of first row are initialized to 0 and the second row to one. The initialization is done row-by-row. The above statement can be equivalently written by surrounding the elements of each row by braces as:

```
int marks[2][3]={{0,0,0},{1,1,1}};
```

2-D ARRAYS (INITIALIZATION)…

- We can also initialize a 2-D array in the form of a matrix as:

```
int marks[2][3]={
                    {0,0,0},
                    {1,1,1}
                };
```

- If the values are missing in an initialization statement, they are automatically set to zero.

  E.g.

```
int marks[2][3]={   {1,1},
                    {2}
                };
```

  Here the first two elements of the first row are initialized to one, the first element of the second row to two, and all other elements are zero.

2-D ARRAYS (INITIALIZATION)…

- When all the elements are to be initialized to zero, the following short-cut method may be used:

  ```
  int marks[3][5]={{0}, {0}, {0}};
  ```

  Here, the elements of each row is explicitly initialized to zero while other elements are automatically initialized to zero.

- Note: First dimension may be empty while initialization of 2-D array.

  E.g.

  ```
  int marks[ ][3]={{2, 4, 6},{8, 10, 12}};
  ```

  is equivalent to

```
marks[0][0]=2;   marks[0][1]=4;   marks[0][2]=6;
marks[1][0]=8;   marks[1][1]=10;  marks[1][2]=12;
```

ACCESSING 2-D ARRAY ELEMENTS

- In a 2-D array declaration, the first dimension specifies no. of rows and second dimension specifies no. of columns.
- Consider an array *marks* of size 4*3 with elements having values:

```
int marks[4][3]={35,10,11,34,90,76,13,8,5,76,4,1};
```

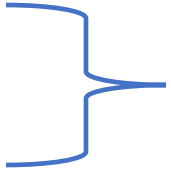
- This array can be realized as a matrix having 4 rows and 3 columns as:

35	10	11
34	90	76
13	8	5
76	4	1

ACCESSING 2-D ARRAY ELEMENTS…

- To access a particular element of a 2-D array, we have to specify the array name, followed by two square brackets with row and column number inside it.
- Thus, marks[0][0] accesses 35, marks[1][1] accesses 90, marks[2][2] accesses 5 and so on.
- Note: Array traversal is row-by-row to access the particular element because "*2-D array can be visualized as an array of 1-D arrays*". Explain?????
- Note: Nested loops are used to traverse the 2-D arrays.

```
//Program to display a matrix on screen
#include <stdio.h>
#include <conio.h>
void main()
{
int matrix[][3]={12,15,18,9,16}, i, j;
clrscr();
printf("\nThe entered matrix is:\n");
   for(i=0;i<2;i++)
            {
            for(j=0;j<3;j++)
                        {
                        printf("%d\t", matrix[i][j]);
                        }
            printf("\n");
            }
getch();
}
```



Nested loop

```
//Program to read a matrix of size M*N from user and display it on screen
#include<stdio.h>
#define M 2
#define N 4
int main()
{
int matrix[M][N], i, j;
printf("\nEnter the matrix elements:\t");
   for(i=0;i<M;i++)  {
            for(j=0;j<N;j++)      {
                     scanf("%d", &matrix[i][j]);
                     }
            }
printf("\nThe entered matrix is:\n");
   for(i=0;i<M;i++) {
            for(j=0;j<N;j++){
                     printf("%d\t", matrix[i][j]);
                     }
            printf("\n");
            }
getch();
Return 0;
}
```

```
/*Program to read two M*N matrices and display their sum/difference*/
#include <stdio.h>
#include <conio.h>
#define M 3
#define N 3
int main()
{
int matrix1[M][N], matrix2[M][N], sum[M][N], i, j;
printf("\nEnter the elements of first matrix:\t");
for(i=0;i<M;i++)
    {
    for(j=0;j<N;j++)
            {
            scanf("%d", &matrix1[i][j]);
            }
    }
printf("\nThe first matrix is:\n");
for(i=0;i<M;i++)
    {
    for(j=0;j<N;j++)
            {
            printf("%d\t", matrix1[i][j]);
            }
    printf("\n");
    }
```

```
printf("\nEnter the elements of second matrix:\t");
for(i=0;i<M;i++)
   {
   for(j=0;j<N;j++)
            {
            scanf("%d",&matrix2[i][j]);
            }
   }

printf("\nThe second matrix is:\n");
for(i=0;i<M;i++)
   {
   for(j=0;j<N;j++)
            {
            printf("%d\t",matrix2[i][j]);
            }
   printf("\n");
   }
```

```
for(i=0;i<M;i++)
    {
    for(j=0;j<N;j++)
                {
                sum[i][j]=matrix1[i][j]+matrix2[i][j];
                }
    }

printf("\nThe sum of the matrices is:\n");
for(i=0;i<M;i++)
    {
    for(j=0;j<N;j++)
                {
                printf("\t%d", sum[i][j]);
                }
    printf("\n");
    }
getch();
return 0;
}
```

```
// Program to find transpose of a matrix
#include <stdio.h>
#include <conio.h>
#define M 3
#define N 3
int main()
{
int matrix[M][N],transpose[N][M],i,j;
printf("\nEnter the elements of matrix:\t");
    for(i=0;i<M;i++) {
                for(j=0;j<N;j++) {
                                scanf("%d", &matrix[i][j]);
                                }
                }
printf("\nThe matrix to be transposed is:\n");
    for(i=0;i<M;i++)        {
                for(j=0;j<N;j++) {
                                printf("%d\t", matrix[i][j]);
                                }
                printf("\n");
                }
```

```
/*finding transpose matrix*/
for(i=0;i<M;i++)            {
    for(j=0;j<N;j++)                    {
                transpose[j][i]=matrix[i][j];
                }
    }

printf("\nThe transpose matrix is:\n");
for(i=0;i<M;i++)            {
    for(j=0;j<N;j++)                    {
                printf("%d\t",transpose[i][j]);
                }
    printf("\n");
    }
getch();
return 0;
}
```

```
//Program to find the sum of squares in a diagonal of a square matrix
#include <stdio.h>
#include <conio.h>
int main()
{
int matrix[10][10],i,j,sum=0,M,N;
printf("\nEnter order of square matrix (less than 10*10):\t");
scanf("%d %d",&M,&N);
    if(M>10 || N>10)
    {
    printf("\nThe order is out of range.\n");
    getch();
    exit();
    }
                if(M!=N)
                {
                printf("Not square matrix.\n");
                getch();
                exit();
                }
```

```
printf("\nEnter the elements of matrix:\t");
for(i=0;i<M;i++)
   {
   for(j=0;j<N;j++)
            {
            scanf("%d",&matrix[i][j]);
            }
   }
printf("\nThe matrix is:\n");
for(i=0;i<M;i++)
   {
   for(j=0;j<N;j++)
            {
            printf("%d\t",matrix[i][j]);
            }
   printf("\n");
   }
```

```
for(i=0;i<M;i++)
   {
   for(j=0;j<N;j++)
            {
            if(i==j)
            sum = sum + matrix[i][j]*matrix[i][j];
            }
   }
printf("\nThe sum of squares of elements on a diagonal is:%d", sum);
getch();
}
```

```
/*Program to compute the product of two matrices if possible*/
#include <stdio.h>
#include <conio.h>
Int main()
{
int matrix1[10][10],matrix2[10][10],i,j,k,product[10][10],M,N,P,Q;
int row_mul_col=0;
printf("\nEnter order of first matrix (less than 10*10):\t");
scanf("%d %d",&M,&N);
printf("\nEnter order of second matrix (less than 10*10):\t");
scanf("%d %d",&P,&Q);
   if(N!=P)
   {
   printf("\nThe matrices are unsuitable for multiplication.\n");
   getch();
   exit();
   }
```

```
printf("\nEnter the elements of first matrix:\t");
for(i=0;i<M;i++)
   {
   for(j=0;j<N;j++)
            {
            scanf("%d",&matrix1[i][j]);
            }
   }
printf("\nThe first matrix is:\n");
for(i=0;i<M;i++)
   {
   for(j=0;j<N;j++)
            {
            printf("%d\t",matrix1[i][j]);
            }
   printf("\n");
   }
```

```
printf("\nEnter the elements of second matrix:\t");
for(i=0;i<P;i++)
   {
   for(j=0;j<Q;j++)
            {
            scanf("%d",&matrix2[i][j]);
            }
   }
printf("\nThe second matrix is:\n");
for(i=0;i<P;i++)
   {
   for(j=0;j<Q;j++)
            {
            printf("%d\t",matrix2[i][j]);
            }
   printf("\n");
   }
```

```
/*multiply two matrices*/
for(i=0;i<M;i++)            {
    for(j=0;j<Q;j++)  {
                for(k=0;k<N;k++)
                            {
                            row_mul_col += matrix1[i][k]*matrix2[k][j];
                            }
                            product[i][j]=row_mul_col;
                            row_mul_col=0;
                }
    }

printf("\nThe matrix after multiplication is:\n");
for(i=0;i<M;i++) {
    for(j=0;j<Q;j++){
                printf("%d\t",product[i][j]);
                }
    printf("\n");
    }
getch();
return 0;
}
```

Thank you !