**CHAPTER – 5**

**ARRAYS**

An array is a fixed-size sequenced collection of elements of the same data type. It is simply a grouping of like-type data. It is used to represent a list of numbers or list of names.

Some of the examples are

* List of employees in an organization.
* List of customers and their telephone numbers.
* List of temperatures recorded every hour in a day.

An array is a sequenced collection of related data items that share a common name. For instance, we can use an array name salary to represent a set of salaries of a group of employees in an organization. We can refer to the individual salaries by writing a number called index or subscript in brackets after the array name.

Ex: salary [10] represent the salary of 10th employee. While the complete set of values is referred to as an array, individual values are called elements.

We can use arrays to represent not only simple lists of values but also tables of data in two, three or more dimensions. Here, we introduce the concept of an array and discuss how to use it to create and apply the following types of arrays

* One-dimensional arrays
* Two-dimensional arrays
* Multidimensional arrays

**One-dimensional arrays:**

A list of items can be given one variable name using only one subscript and such a variable is called a single-subscripted variable or a one-dimensional array.

If we want to represent a set of five numbers, say {69,32,84,20,41}, by an array variable number, then we may declare the variable number as follows:

Int number[5];

And the computer reserves five storage locations as shown below:

|  |
| --- |
|  |
|  |
|  |
|  |
|  |

number[0]

number[1]

number[2]

number[3]

number[4]

The values of the array elements can be assigned as follows:

number[0] = 69;

number[1] = 32;

number[2] = 84;

number[3] = 20;

number[4] = 41;

This would cause the array number to store the values as shown below:

And the computer reserves five storage locations as shown below:

|  |
| --- |
|  |
|  |
|  |
|  |
|  |

number[0]

number[1]

number[2]

number[3]

number[4]  
These elements may be used in programs just like any other C variable. The subscript of an array can be integer constants, integer variables like j, or expressions that yield integers.

**Declaration of one-dimensional arrays :**

Like another variable, arrays must be declared before they are used so that the compiler can allocate space for them in memory. The general form of array declaration is

type variable\_name[size];

The type specifies the type of elements that will be contained in the array, such as int, float or char and the size indicates the maximum number of elements that can be stored inside the array.

Ex: float height[50];

declares the height to be an array containing 50 real elements. Any subscripts 0 to 49 are valid. Similarly int group[10];

declares the group as an array to contain a maximum of 10 integer constants.

* Any reference to the arrays outside the declared limits would not necessarily cause an error. Rather , it might result in unpredictable program results.
* The size should be either a numeric constant or a symbolic constant.

The C language treats character strings simply as arrays of character. The size in a character string represents the maximum number of characters that the string can hold.

i e., char name[10];

declares the name as a character array (string) variable that can hold a maximum of 10 characters. Suppose we read the following string constant into the string variable name.

“ HELLO WORLD”

Each character of the string is treated as an element of the array name and is stored in memory as follow:

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| ‘H’ | ‘E’ | ‘L’ | ‘L’ | ‘O’ | ‘’ | ‘W’ | ‘O’ | ‘R’ | ‘L’ | ‘D’ | ‘\0’ |

When the compiler sees a character string, it terminates it with an additional null character. Thus, the element name[10] holds the null character ‘\0’. When declaring character arrays, we must allow one extra element space for the null terminator.

**Accessing Elements in Arrays:**

C uses an index to access individual elements in an array. The index must be an integral value or an expression that evaluates to an integral value. The simplest form for accessing an element is a numeric constant.

**Initialization of one-dimensional arrays :**

An array can be initialized at either of the following stages:

* + At compile time
  + At run time

**Compile Time Initialization:**

We can initialize the elements of arrays in the same way as the ordinary variables when they are declared. The general form of initialization of arrays is :

**type array-name[size]={list of values};**

The values in the list are separated by commas.

Ex: int number [3] = {0,0,0 };

The above statement will declare the variable number as an array of size 3 and will assign zero to each element. If the number of values in the list is less than the number of elements, then only that many elements will be initialized. The remaining elements will be set to zero automatically.

Ex: float total [5] = {0.0,15.75,-10};

Will initialize the first three elements to 0.0, 15.75 and -10.0 and the remaining two elements to zero.

The size may be omitted. In such cases, the compiler allocates enough space for all initialized elements.

Ex: int counter [ ] = {1, 1, 1, 1};

Will declare the counter array to contain four elements with initial values 1.

Character arrays may be initialized in a similar manner.

Ex: char name [ ] = { ‘J’,’o’,’h’,’n’,’\0’};

Declares the name to be an array of five characters, initialized with the string “John” ending with the null character. Alternatively, we can assign the string directly as follows:

Char name [ ] = “John”;

Ex: int number [5] = {10, 20};

The above statement will initialize the first two elements to 10 and 20 respectively, and the remaining elements to 0.

The statement char city [5] = {‘B’};

will initialize the first element to ‘B’ and the remaining four to NULL.

**Run Time Initialization:**

An array can be explicitly initialized at run time. We this approach for initializing large arrays.

Ex: --------------

--------------

for (i=0;i<100;i++)

{

If i < 50

Sum[i] = 0.0; /\* assignment statement \*/

Else

Sum[i] = 1.0;

}

---------------

---------------

We can also use a read function such as scanf to initialize an array.

Ex: int x [3];

Scanf (“%d%d%d”,&x[0],&x[1],&x[2]);

Will initialize array elements with the values entered through the keyboard.

**Two-dimensional arrays:**

**Definition:**

Grouping of variables of the same type under a single name with two dimensions. When processed, this grouping of variables produces results that can be displayed in columns and rows.

C allows us to define a table of items by using two-dimensional arrays.

Two-dimensional arrays are defined as follows:

type array\_name[row\_size][column\_size];

Two-dimensional arrays are stored in arrays as follows:

Column 0 column 1 column 2

[0][0] [0][1] [0][2]

|  |  |  |
| --- | --- | --- |
| 310 | 275 | 365 |

Row 0

[1][0] [1][1] [1][2]

|  |  |  |
| --- | --- | --- |
| 10 | 190 | 325 |

Row 1

[2][0] [2][1] [2][2]

|  |  |  |
| --- | --- | --- |
| 405 | 235 | 240 |

Row 2

[3][0] [3][2] [3][2]

|  |  |  |
| --- | --- | --- |
| 310 | 275 | 365 |

Row 3

As in the single dimensional arrays, each dimension of the array is indexed from zero to its maximum size minus one; the first index selects the row and the second index selects the column within the row.

**Initialization of two-dimensional arrays :**

Two-dimensional array may be initialized by following their declaration with a list of initial values enclosed in braces.

Ex: int table[2][3] = {0, 0, 0, 1, 1, 1};

initializes the elements of the first row to zero and the second row to one. The initialization is done by row by row. We can write the above statement as:

Ex: int table [2][3] = {{0,0,0}, {1,1,1}};

by surrounding the elements of each row by braces.

We can also initialize the two dimensional array in the form of a matrix as shown below:

Ex: int table [ ][3] = {

{0, 0, 0},

{1, 1, 1}

}; is allowed.

If the values are missing in an initializer, then they are automatically set to zero.

Ex: int table [2][3] = {

{1, 1},

{2}

};

will initialize the first two elements of the first row to one, the first element of the second row to two, and all other elements to zero.

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

Ex: int m[3][5] = { {0}, {0}, {0}}; or int m[3][5] = { 0, 0, 0};

Ex: Program for matrix multiplication, addition and subtraction.

#include<stdio.h>

#include<conio.h>

void main()

{

int i,j,k,ch,r1,r2,c1,c2,a[20][20],b[20][20],c[20][20];

clrscr();

printf("Enter the order of the 1st matrix :: \n");

scanf("%d %d",&r1,&c1);

printf("Enter the order of the 2nd matrix ::\n");

scanf("%d %d",&r2,&c2);

printf("Enter the elements in the 1st matrix ::\n ");

for(i=0;i<r1;i++)

{

for(j=0;j<c1;j++)

scanf("%4d",&a[i][j]);

printf("\n");

}

printf("Enter the elements in the 2nd matrix ::\n");

for(i=0;i<r2;i++)

for(j=0;j<c2;j++)

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

do

{

printf("\*\*\*\*\*\*\*\*\*\*\*\*MENU\*\*\*\*\*\*\*\*\*\*\*");

printf("\n1.MATRIX ADDITION\n");

printf("2.MATRIX SUBTRACTION\n");

printf("3.MATRIX MULTIPLICATION\n");

printf("4.EXIT\n\n\n");

printf("Enter ur choice");

scanf("%d",&ch);

switch(ch)

{

case 1: if((r1==r2)&&(c1==c2))

{

for(i=0;i<r1;i++)

{

for(j=0;j<c1;j++)

{

c[i][j]=a[i][j]+b[i][j];

printf("%3d",c[i][j]);

}

printf("\n");

} }

else printf("matrix addition is not possible");

break;

case 2:

if((r1==r2)&&(c1==c2))

{

for(i=0;i<r2;i++)

{

for(j=0;j<c2;j++)

{

c[i][j]=a[i][j]-b[i][j];

printf("%3d",c[i][j]);

}

printf("\n");

} }

else printf("matrix subtraction is not possible");

break;

case 3: if(c1==r2)

{

for(i=0;i<r1;i++)

{

for(j=0;j<r1;j++)

{

c[i][j]=0;

for(k=0;k<r2; k++)

{

c[i][j]+=a[i][k]\*b[k][j];

}

printf("%4d",c[i][j]);

}

printf("\n");

} }

else printf("matrix multiplication is not possible");

break;

case 4: exit(0);

} }

while(ch<=4);

getch();

}

**Output:-**

enter r1 & c1 3 3

enter r2 & c2 3 3

Enter the values for a 1 2 3 4 5 6 7 8 9

Enter the values for b 4 5 6 7 8 9 1 2 3

\*\*\*\*\*\*\*\*\*\*\*\*MENU\*\*\*\*\*\*\*\*\*\*\*

1.MATRIX ADDITION

2.MATRIX SUBTRACTION

3.MATRIX MULTIPLICATION

4.EXIT

Enter ur choice1

5 7 9

11 13 15

8 10 12

**Multi-dimensional arrays:**

C allows array of three or more dimensions. The exact limit is determined by te compiler.The general form of a multi-dimensional array is :

Type array\_name [s1][s2][s3]......[sm];

Where si the size of ith dimension.

Ex: int survey [3][5][12];

Where survey is a three-dimensional array declared to contain 180(3\*5\*12) integer type elements.

ANSI C does not satisfy any limit for array dimension. However, most compilers permit seven to ten dimensions. Some allow even more.

**Dynamic array:**

An array can be created at compile time by specifying size in the source code has a fixed size and cannot be modified at run time. The process of allocating memory at compile time is known as static memory allocation and the arrays that receive static memory allocation are called static arrays.

The process of allocating memory at run time is known as dynamic memory allocation and the arrays created at run time are called dynamic arrays. Dynamic arrays are created using pointer variables and memory management functions malloc, calloc and realloc. These functions are included in the header file <stdlib.h>.

**Applications of Arrays:**

* Arrays are used to Store List of values.
* Arrays are used to Perform Matrix Operations
* Arrays are used to implement Search Algorithms
* Arrays are used to implement Sorting Algorithms
* Arrays are used to implement Datastructures
* Arrays are also used to implement CPU Scheduling Algorithms