# 16

# ARRAYS

## 16.1 INTRODUCTION

As we stated earlier that the variables are the entities in 'C' which are used to hold data in memory. But the concept of variables does not solve the indeterminate number of values is to be stored and operated upon. In case of storing say 100 values, it is a very difficult task to store 100 different variable names with their values. Arrays are the solution to this problem. Arrays are nothing but a single name to a whole group of similar data. The position in terms of arrays is known as subscript. Each subscript must be expressed as a non-negative integer.

| y[0] | y[1] | y[2] | y[3] |
|---|---|---|---|

The number of subscripts determines the dimension of the array.

## 16.2 OBJECTIVES

After going through this lesson you will be able in a position to

- define AN ARRAY
- process AN ARRAY
- pass ARRAYS TO FUNCTIONS
- process MULTIDEMNSIONAL ARRAYS AND STRINGS.

### 16.3 DEFINING AN ARRAY

Arrays are defined in much the same manner as ordinary variables except that each array name must be followed by a size specification. For a one-dimensional array, the size is specified by a positive integer expression enclosed in square brackets.

For example

Storage-class data-type array name[expression];

```
int a[10];
char text[100];
static char text[100];
```

The array's size can be defined in terms of a symbolic constant rather than a fixed integer quantity.

The following example reads a one-dimensional character array. and convert it to uppercase and then print it.

```
# include <stdio.h>
main()
{
char str[20];
int i;
printf("Enter any line of 9 characters");
scanf("\s", str);
scan("%s", str);
for(i=0;, i<20; ++i)
{
putchar(toupper(str[i] ));
}
}
```

Automatic arrays, unlike automatic variables, cannot be initialized. But the definitions of external and static arrays can include the assignment of initial values if required. The initial values must appear in the order in which they will be assigned to the individual array elements, enclosed in braces and separated by commas. The general

form is

storage class data-type array. name[expression]={value1, value2 - - - , value n};

For example, char school[4] ={'O','P','E','N'};

```
int marks[10]={1,2,3,4,5,6,7,8,9,10};
static float y[3]={o,0.3,0.2};
```

All individual array elements that are not assigned explicit initial values will automatically be set to zero. This includes the remaining elements of an array in which certain element have been assigned non zero values.

The array size need not be specified explicitly when initial values are included as a part of an array definition. With a numerical array, the array size will automatically be set equal to the number of initial values included within the definition.

For example

```
int array[]={4,8,3,7,5};
```

Since the square brackets following the array name are empty, the compiler determines how many elements to allocate for the array by counting the number. of values within the curly braces. This approach can help to avoid errors. If the dimension is specified explicitly, and the curly braces contain more initialization values than are needed, a syntax error is flagged by the compiler.

The case of strings are different. The array size specification in this case is usually omitted. The proper array size will be assigned automatically. This will include a provision for the null character.

If a program requires a one-dimensional array declaration the declaration is written in the same manner as the array definition with the following exception.

1. The square brackets may be empty, since the array size will have been specified as a part of the array definition. Array declaration are customarily written in this form.

2. Initial values cannot be included in the declaration. Following are the examples of defining an External array.

```
int a[]= { 1,2,0};              /*external array definition */
char text []="open";            /*external array definition  */
extern void dummy(void);        /* external function declaration */
```

## 16.4 PROCESSING AN ARRAY

Single operations involving entire array are not permitted in 'C'. If two arrays are same in every respect then different operations must be carried out on an element by element basis. This usually accomplished within a loop, where each pass through the loop is used to process one array element. The number of passes through the loop will therefore equal the number of array elements to be processed.

Let us consider an example of processing an array.

```
#include <stdio.h>
main()
{
int number,i;
float avg, sum=0;
float arr[20];
printf("How many numbers will be averaged");
scanf("%d", & number);
printf("\n");
for (i=0; i<number; ++i);
printf ("Enter number")
{
scanf("%f", & arr[i]);
sum +=arr[i];
}
avg= sum/number;
printf("\n The average is %f\n", avg);
}
```

Here, each value gets entered in the array "arr" one-by-one so arr [0] element will fill first then arr[1] - - and so on.

## 16.5 PASSING ARRAYS TO FUNCTIONS

An array name can be used as an argument to a function, thus permitting the entire array to be passed to the function.

To pass an array to a function, the array name must appear by itself, without brackets or subscripts, as an actual argument or parameter within the function call. The corresponding formal parameter is written in the same manner, though it must be declared as an array within the formal argument declarations. When declaring a one-dimensional array as a formal argument, the array name is written with a pair of empty square brackets. The size of the array is not specified within the formal argument declaration.

Let us consider an example of passing arrays to a function

```
# include <stdio.h>
main()
{
int n, i;
float avg;
float avr[20]; /*array definition */
float average();/* function declaration */
printf("Enter number");
scanf("%d", &n);
printf("\n");
for (i=0; i<n; ++i)
{
printf("Enter number");
scanf("%f",& arr[i]);
}
        avg=average (n,arr);
        printf("The average of numbers is %f", avg);
}
        float average(a,x)   /*function DEFINITION */
        int a;                  /*formal argument declaration */
        float x[ ];             /* formal argument (array) declaration*)
{
    float sum ; int i=0;
    float avg=0;
    for (i=0; i<a; ++i)
    {
```

```
        sum+=x[i];
}
        avg=sum/a;
        return avg;
}
```

Within main there is a call to the function average. This function call contains two actual argument—the integer variable n, and the one dimensional, floating-point array arr. The arr appear as an ordinary variable, within the function call.

In the first line of the function definition, there are 2 formal arguments, called a and x.

When an array is passed to a function, however, the values of the array element are not passed to the function. But the array name is interpreted as the address of the first array element. This address is assigned to the corresponding formal argument when the function is called. The formal argument therefore becomes a pointer to the first array element. Arguments passed in this manner are said to be passed by reference rather than by value. When a reference is made to an array element within the function , the value of the element's subscript is added to the value of the pointer to indicate the address of the specified array element. Therefore, any array element can be accessed from within the function . If an array element is altered within the function., the alteration will be recoganised in the calling portion of the program.

With the return statement array cannot be used. If the elements of an array are to be passed back to the calling portion of the program, the array must either be defined as an external array or it must be passed to the function as a formal argument.

**INTEXT QUESTIONS**

1. What is an array ?
2. How is the integer array x, containing 50 elements, declared in 'C'?
3. What is the subscript of the first element of an array in 'C' ?.
4. What are the rules for naming array ?

5. Is it possible to declare and initialize an array in 'C' simultaneously ? If so, how ?

6. Must the elements of an array be read in or printed out in order of subscript ?

## 16.6 MULTIDIMENSIONAL ARRAYS

The arrays we have used so far have been one dimensional. The elements of the array could be represented either as a single column or as a single row.

A two dimensional array is a grid containing rows and columns, in which each element is uniquely specified by means of its row and column coordinates. Multidimensional arrays are defined in much the same manner as one dimensional arrays, except that a separate pair of square brackets is required for each subscript. Thus a two-dimensional array will require two pairs of square brackets, a three dimensional array will require three pairs of square brackets and so on. A multidimensional array definition can be written as

storage-class data-type array-name[expression1][expression 2] ——[expression n];

The two dimensional array is like a matrix where position of each element is specified by both the column and row numbers.

For example:

Matrix x

| | | 0 | 1 | 2 | 3 | Column |
|---|---|---|---|---|---|---|
| Row | 0 | 4 | 5 | 1 | 0 | |
| | 1 | 2 | 6 | 9 | 1 | |
| | 2 | 5 | 2 | 6 | 7 | |

Suppose if want to locate '9' then its specification is x[1][2]

The row subscript generally is specified before the column subscript. In C, each subscript be written within its own separate pair of brackets.

For example: Definition of Multidimensional arrays are given below.

float table[10][10];

char string[10]20];

If a multidimensional array definition include the assignment of initial values , then care must be given to the order in which the initial values are assigned to the array element. The rule is that the last subscript increases most rapidly, and the first subscript increases least rapidly. Thus, the elements of a two dimensional array will be assigned by rows, that is , the elements of the first row will be assigned, then the elements of the second row, and so on.

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

Thus sample [0][0]=1 sample[0][1]=2 sample [0][2]=3 sample [0][3]=4 sample[1][0]=5

sample[1][1]=6 sample[1][2]=7 sample[1][3]=8 sample[2][0]=9 sample[2][1]=10 sample[2][2]=11 sample[2] [3]=12

You can write another definition for array sample as given below:

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

If the definition is given like this

```
int sample[3][4]={
{1,2,3},
{4,5,6},
{7,8,9}
};
```

Then this definition assigns values only to the first three elements in each row. Rest last 3 element of each row gets a value of zero.

If the above definition is written in another way like

int sample [3][4]={1,2,3,4,5,6,7,8,9};

then here also 3 elements will be assigned zeros, but the order of the assignments will be different. In this case the last 3 elements will get a value zero.

Multidimensional arrays are processed in the same manner as one dimensional arrays, on an element-by-element basis.

Let us understand the example of multidimensional arrays of adding two tables of numbers.

```
#include <stdio.h>
main()
{
int rows, cols;
int a [10][20], b [10][20], c [10][20],
void read (int a [ ][20], int rows, int cols);
void sum (int a [][20], int b [ ][20], int c [ ][20], int rows, int-
cols);
void write( int c [ ][20], int rows, int cols);
printf( “Enter number of rows and columns”);
scanf(“%d %d”, & rows, & cols);
printf(“\n” First Table” );
read (a,rows, cols);
printf(“\n\n Second table” );
read(b, rows, cols);
sum(a,b,c, rows, cols);
write(c, rows, cols);
}
void read(int a[ ] [20], int m, int n)
{
int row, col;
for(row-0; row<m; ++ row)
{
for(col=0; col<n; ++ cols)
scanf(“%d”, & a[row][col]);
}
return;
```

```
}
void sum (int a[ ][20], int b[ ] [20], int c[ ] [20], int m, int n)
{
int row, col;
for(row=0; row<m; ++ row)
for (col=0; col<n; ++ col)
c[row][col]=a[row][col]+b[row][col];
return;
}
void write(int a[ ][20], int m, int n)
{
int row, col;
for(row=0; row<m; ++row){
for(col=0; col<n; ++col)
printf("%d", a[row][col]);
printf("\n");
}
return;
}
```

**INTEXT QUESTIONS**

7. How are multidimensional arrays defined ?

8. If all the elements of a two-dimensional array are initialized in the declaration of the array both subscripts may be omitted , is it is true or false ?

## 16.7. ARRAYS AND STRINGS

Strings can be represented as a one-dimensional character-type array. Each character within the string will be stored within one element of the array. For example,

char name[ ]= {'N', 'A', 'T','I','O','N','A','L','\0'};

Each character in the array occupies one byte of memory and the last character is always '\0'. The terminating null is important, because it is the only way the functions that work with a string can

know where the string ends. In facts, a string which is not terminated by a '\0' is not really a string but merely a collection of characters. The above string can be also initialized as

char name[ ]= "National";

In this declaration '\0' is not necessary. C inserts the null character automatically.

Let us consider an example to demonstrate printing of string

```
#  include <stdio.h>
main()
{
char name [ ]="National";
int i= 0;
while(i<=8)
{
printf("%c", name [i];
i ++;
}
}
```

The output is **National.**

With the help of null character this program can be written in the following manner:

```
#include <stdio.h>
main()
{
char name [ ]=**"National";**
int i=0;
while (name [i]!='\0')
{
printf ("%c", name[i ] );
i++;
}
}
```

printf() does not print the '\0'. The %s used in printf() is a format specification for printing out a string.

```
#include<stdio.h>
main()
{
char name[25];
printf("Enter your name");
scanf("%s", name);
printf("%s", name);
}
```

It will print the same string as entered from the user. The length of string should not exceed the dimension of the character array. This is because the C compiler doesn't perform bounds checking on character arrays. scanf() is not capable of receiving multi word strings. The way to get around this limitation is by using the function gets (). The example is shown below:

```
#include <stdio.h>
main()
{
char name [25];
printf("Enter your full name");
gets(name);
puts(name);

}
```

Following is an example to print the length of string using gets and null character.

```
# include <stdio.h>
main()
{
int i;
char name [30];
printf("enter some string");
gets(name);
for(i=0; name [i]!= '\0'; i++);
```

```
printf("The length of the string is %d", i);
}
```

The length of the string can be calculated with the help of standard library function strlen().

```
#include <stdio.h>
main()
{
int i;
char name [30];
printf("enter some string");
gets(name);
i=strlen(name);
printf("The length of the string is %d",i);

}
```

There are strcpy(), strcmp, strcat() standard library functions all of these have different usuages.

```
# include <stdio.h>
main()
{
char name [ ]="National";
char name1[20];
strcpy(name1, name);
printf("\n Original string=%s", name);
printf("\n Copied  string =%s", name1);
}
```

output is : **Original string=National**

**Copied string= National**

strcpy() function copies one string to another.

Similarly, strcmp() function compares two strings to find out whether they are same or different. The two strings are compared character by character until there is a mismatch or end of one of the strings is reached, whichever occurs first. If the two strings are identical, strcmp() returns a value zero. If they are not, it returns the numeric

difference between the ASCII values of the non-matching character.

```
#include <stdio.h>
main()
{
char name[ ]="National";
char name1[ ]= "Open";
int i, j, k;
i=strcmp(name,"National");
j=strcmp(name, name1);
k= strcmp(name, "National School");
printf("\n %d%d%d", i,j,k);
}
```

The output is as follows:

0,1,-32

In the first call to strcmp(), the 2 strings are identical, "National" and "National" and the value returned by strcmp() is zero. In the second call, the first character of "National", does not match with first character of "Open" and the result is 1, which is the numeric difference between ASCII value of N and 0. In the third call to strcmp() "National" does ASCII not match with National school, because the null character at the end of National does not match the blank in National school. The values returned is –32, which is the value of null character minus the ASCII value of space i.e '\0' minus ' ', which is equal to –32.

You can do rest of the standard library functions by yourself.

**Two dimensional Array of characters**.

Let us first consider an example of 2D array of character.

```
#include <stdio.h>
main()
{
char sample[6][10]= {
"Vaishali"
```

```
"Akanksha"
"Shivansh"
"Tanishq"
"Anmol"
"Yash"
};
int i, flag, a;
char name[10];
printf("Enter your name");
scanf("%s", name);
flag=False;
for(i=0, i<5;i++)
{
a = strcmp(&sample[i][0], name);
if(a= = 0)
{
printf("You are a right candidate");
flag =True;
break;
}
}
if(flag= =False)
printf("Wrong candidate");
}
```

In this program note that how the two dimensional character array has been initialised. The order of the subscripts in the array declaration is important. The first subscript gives the number of names in the array, while the second subscript gives the length of each item in the array.

The function strcmp() is also used to compare two strings which will give the value 0 if strings are equal. The names would be stored in the memory as shown in fig. Each string ends with '\0'. The arrangement is somewhat similar to that of a two dimensional numeric array.

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 1001 | V | a | i | s | h | a | l | i | \0 | |
| 1011 | A | k | a | n | k | s | h | a | \0 | |
| 1021 | S | h | i | v | a | n | s | h | \0 | |
| 1031 | T | a | n | i | s | h | q | \0 | | |
| 1041 | A | n | m | o | l | \0 | | | | 1060 (last location) |
| 1051 | Y | a | s | h | \0 | | | | | |

Two dimensional character Array.

As scan from the above pattern some of the names do not occupy all the bytes reserved for them. For example, even though 10 bytes are reserved for storing the name "Vaishali", it occupies only 9 bytes. Thus 1 byte go waste. Similarly, for each name there is some amount of wastage. This wastage of memory can be avoided using arrays of pointers which we will discuss in next chapter.

## 16.8 WHAT YOU HAVE LEARNT

In this lesson you have learnt about arrays, what are one-dimensional array and two dimensional arrays. Now you can easily pass an array to any function. You have also learnt about multidimensional arrays and strings.

## 16.9 TERMINAL QUESTIONS

1. When passing an array to a function, how must the array argument be written?
2. How can a list of strings be stored within a two dimensional array ?
3. Write a program to compare two strings entered by the user without help of strcmp() function.
4. Write a program to copy a string without help of strcpy() function.
5. Write a program to enter strings from user and then sort them in alphabetical order.(You can use string standard library functions)
6. Write a program to count the number of vowels in a string entered from the user.
7. Write a program to convert a string from lowercase to uppercase.
8. Write a program to join two strings e.g. Suneeta, Gupta.

## 16.10 KEY TO IN-TEXT QUESTIONS

1. Array is an ordered collection of elements that share the same name.

2. int x[50];

3. Zero

4. The same as for naming regular variables or functions. An array cannot have the same name as a variable or function within the same program.

5. Yes, the array declaration is followed immediately by an equal sign. This is followed by the list of values to be assigned enclosed in braces.

6. No, the elements of an array may be accessed in any order at all.

7. Storage-class data type array name[expression 1][expression 2] - - — [expression n];

8. False, only the first (row) subscript can be omitted, with first pair of square brackets left empty. The second subscript must always be explicitly specified.