MANGALAYATAN UNIVERSITY
Learn Today to Lead Tomorrow
NAAC GRADE A+ Accredited University

BACHELOR OF COMPUTER APPLICATION (BCA)

CSB-1112 (Problem Solving Using C)

Block 2: Arrays and Strings

Unit 5: Arrays in C, Introduction to Arrays

Structure	
5.0	Introduction
5.1	Objectives
5.2	Array Declaration
5.3	Array Initialization
5.4	Sub Script
5.5	Passing Arrays to Functions
5.6	Processing the Arrays
5.7	Multi-Dimensional Arrays
5.8	Array of Strings
5.9	Conclusion
5.10	Questions
5.11	References

5.0 Introduction

In this Unit, arrays serve as fundamental data structures, allowing developers to store and manipulate collections of elements efficiently. In C, arrays are essential constructs that provide a systematic way to organize and access data elements of the same type sequentially. An array in C is a contiguous block of memory locations, each holding an element of the same data type, indexed by a numeric value. This numeric index enables quick and direct access to individual elements, facilitating various operations such as traversal, modification, and retrieval.

Arrays in C offer a versatile and compact means of handling data, making them invaluable in a wide range of applications. Whether storing integers, characters, or user-defined data types, arrays provide a uniform and efficient mechanism for managing collections of elements. Understanding the principles of arrays in C is foundational for mastering the language and building more complex data structures and algorithms.

In this introduction to arrays in C, we will explore the fundamentals of array declaration, initialization, indexing, and manipulation. By grasping these concepts, programmers can harness the power of arrays to streamline their code, optimize memory usage, and tackle a diverse array of programming tasks effectively. Through practical examples and illustrations, we will delve into the intricacies of working with

arrays in C, equipping learners with the knowledge and skills needed to leverage this essential data structure in their programming endeavors.

5.1 Objectives

After completing this unit, you will be able to understand,

- To introduce learners to the fundamental concepts of arrays in C, including declaration, initialization, and accessing array elements using indexing.
- To ensure that learners comprehend the concept of contiguous memory allocation and the role of indexing in array manipulation.
- To provide practical examples and exercises for learners to practice array manipulation techniques and reinforce their understanding.
- To emphasize the importance of memory management practices to avoid memory leaks and optimize program efficiency.
- To highlight the limitations of arrays in C, such as fixed size and lack of built-in bounds checking, and introduce strategies for handling these limitations.

5.2 Array Declaration

Arrays a kind of data structure that can store a fixed-size sequential collection of elements of the same type. An array is used to store a collection of data, but it is often more useful to think of an array as a collection of variables of the same type.

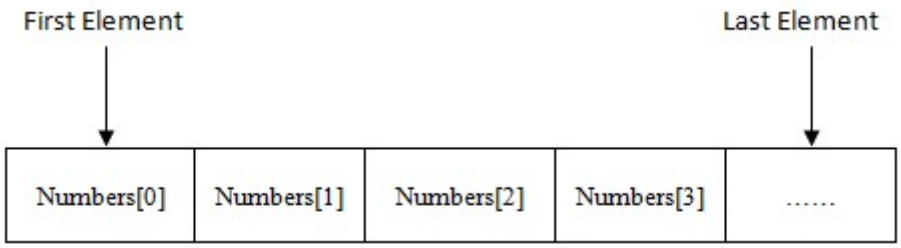
Instead of declaring individual variables, such as number 0, number1, ..., and number 99, you declare one array variable such as numbers and use numbers[0], numbers[1], and ..., numbers[99] to represent individual variables. A specific element in an array is accessed by an index.

All arrays consist of contiguous memory locations. The lowest address corresponds to the first element and the highest address to the last element.

The characteristic features of an array.

- Array is a data structure storing a group of elements, all of which are of the same data type.
- All the elements of an array share the same name, and they are distinguished from one another with the help of an index.
- Random access to every element using a numeric index (subscript).
- A simple data structure, used for decades, which is extremely useful.
- Abstract Data type list is frequently associated with the array data structure.

The declaration of an array is just like any variable declaration with additional size part, indicating the number of elements of the array.



Syntax of Array Declaration

To declare an array in C, a programmer specifies the type of the elements and the number of elements required by an array as follows:

data_type arrayName [arraySize];

This is called a single-dimensional array. The arraySize must be an integer constant greater than zero and type can be any valid C data type.

The following are some of declarations for arrays:

```
int char [80];
float farr [500];
static int iarr [80];
char charray [40];
```

There are two restrictions for using arrays in C:

- The amount of storage for a declared array has to be specified at compile time before execution. This means that an array has a fixed size.
- The data type of an array applies uniformly to all the elements; for this reason, an array is called a homogeneous data structure.

Size Specification

The size of an array should be declared using symbolic constant rather a fixed integer quantity. The use of a symbolic constant makes it easier to modify a program that uses an array. All reference to maximize the array size can be altered simply by changing the value of the symbolic constant.

To declare size as 50 use the following symbolic constant, SIZE, defined:

```
#define SIZE 50
```

This example shows how to declare and read values in an array to store marks of the students of a class.

Example:

Write a program to declare and read values in an array and display them.

```
/* Program to read values in an array*/
# include < stdio.h >
# define SIZE 5                 /* SIZE is a symbolic constant */
main ( )
{
int i = 0;                      /* Loop variable */
int stud_marks[SIZE]; /* array declaration */
/* enter the values of the elements */
for( i = 0;i<size;i++)
{
printf (“Element no. =%d”,i+1);
printf(“ Enter the value of the element:”);
scanf(“%d”,&stud_marks[i]); } printf(“\nFollowing are the values stored in the corresponding array
elements: \n\n”); for( i = 0; i<size;i++)
{
printf(“Value stored in a[%d] is %d\n”i, stud_marks[i]);
```

```
}
}
```

OUTPUT:

Element no. = 1 Enter the value of the element = 11

Element no. = 2 Enter the value of the element = 12

Element no. = 3 Enter the value of the element = 13

Element no. = 4 Enter the value of the element = 14

Element no. = 5 Enter the value of the element = 15

Following are the values stored in the corresponding array elements:

Value stored in a[0] is 11

Value stored in a[1] is 12

Value stored in a[2] is 13

Value stored in a[3] is 14

Value stored in a[4] is 15

Accessing Array Elements

An element is accessed by indexing the array name. This is done by placing the index of the element within square brackets after the name of the array.

Example:

double salary = balance[9];

The above statement will take the 10th element from the array and assign the value to salary variable.

Example:

To use concepts viz. declaration, assignment, and accessing arrays:

```
#include<stdio.h>
int main ()
{
int n[ 10 ];                    /* n is an array of 10 integers */
int i,j;                        /* initialize elements of array n to 0 */
for ( i = 0; i < 10; i++ ) { n[ i ] = i + 100; /* set element at location i to i + 100 */
}
/* output each array element's value */
for (j = 0; j < 10; j++ )
{
printf("Element[%d] = %d\n", j, n[j] );
}
return 0;
}
```

The following Result:

Element[0] = 100

Element[1] = 101

Element[2] = 102

Element[3] = 103

Element[4] = 104

Element[5] = 105

Element[6] = 106

Element[7] = 107

Element[8] = 108

Element[9] = 109

5.3 Array Initialization

Arrays can be initialized at the time of declaration. The initial values must appear in the order in which they will be assigned to the individual array elements, enclosed within the braces and separated by commas.

Initialization of Array Elements in the Declaration

The values are assigned to individual array elements enclosed within the braces and separated by comma.

Syntax of array initialization is:

data type array-name [size] = {val 1, val 2, ...val n};

val 1 is the value for the first array element, val 2 is the value for the second element, and val n is the value for the n array element.

Note: When initializing the values at the time of declaration, then there is no need to specify the size. Let the given example:

```
int digits [10] = {1,2,3,4,5,6,7,8,9,10};
int digits[ ] = {1,2,3,4,5,6,7,8,9,10};
int vector[5] = {12,-2,33,21,13};
float temperature[10] ={ 31.2, 22.3, 41.4, 33.2, 23.3, 32.3, 41.1, 10.8, 11.3, 42.3};
double width[ ] = { 17.33333456, -1.212121213, 222.191345 };
int height[ 10 ] = { 60, 70, 68, 72, 68 };
```

Character Array Initialization

The array of characters is implemented as strings in C. Strings are handled differently as far as initialization is concerned. A special character called null character ‘ \0 ’, implicitly suffixes every string. When the external or static string character array is assigned a string constant, the size specification is usually omitted and is automatically assigned; it will include the ‘\0’character, added at end.

Example:

```
char thing [ 3 ] = “TIN”;
char thing [   ] = “TIN”;
```

The above two statements the assignments are done differently. The first statement is not a string but simply an array storing three characters ‘T’, ‘I’ and ‘N’ and is same as writing: char thing [3] = {‘T’, ‘I’, ‘N’}; whereas, the second one is a four character string TIN\0.

5.4 Sub Script

To the individual element in an array, a subscript is used. to the statement used in

Example:

scanf (“ % d”, &stud_marks[i]);

Subscript is an integer type constant or variable name whose value ranges from 0 to SIZE - 1 where SIZE is the total number of elements in the array.

To individual elements of an array of size 5: Consider the following declarations: char country[ ] = “India”; int stud[ ] = {1, 2, 3, 4, 5}; Here both arrays are of size 5. This is because the country is a char array and initialized by a string constant “India” and every string constant is terminated by a null character ‘\0’.

Example:

```
/* Program to find the maximum marks among the marks of 10 students*/
# include < stdio.h >
# define SIZE 10                /* SIZE is a symbolic constant */
main ( )
{
int i = 0;
int max = 0;
int stud_marks[SIZE];                    /* array declaration */
/* enter the values of the elements */
for( i = 0;i<size;i++)
{
printf(“Student no.=%d”,i+1);
printf(“ Enter the marks out of 50:”);
scanf(“%d”,&stud_marks[i]);
}
/* find maximum */
for( i = 0;i<size;i++)
{
If(stud_marks[i]>max)
max = stud_marks[ i ];
}
printf(“\n\nThe maximum of the marks obtained among all the 10 students is: %d ”,max); }
```

OUTPUT:

Student no. = 1 Enter the marks out of 50: 10

Student no. = 2 Enter the marks out of 50: 17

Student no. = 3 Enter the marks out of 50: 23

Student no. = 4 Enter the marks out of 50: 40

Student no. = 5 Enter the marks out of 50: 49

Student no. = 6 Enter the marks out of 50: 34

Student no. = 7 Enter the marks out of 50: 37

Student no. = 8 Enter the marks out of 50: 16

Student no. = 9 Enter the marks out of 50: 08

Student no. = 10 Enter the marks out of 50: 37

5.5 Passing Arrays to Functions

To pass a single-dimension array as an argument in a function, you would have to declare a formal parameter in one of following three ways and all three declaration methods produce similar results because each tells the compiler that an integer pointer is going to be received. Similarly, you can pass multi-dimensional arrays as formal parameters.

Way-1

Formal parameters as a pointer:

```
void myFunction(int *param)
{
 .
.
.
}
```

Way-2

Formal parameters as a sized array:

```
void myFunction(int param[10])
{
.
.
.
}
```

Way-3

Formal parameters as an unsized array:

```
void myFunction(int param[])
{
.
.
.
}
```

Example:

Now, consider the following function, which takes an array as an argument along with another argument and based on the passed arguments, it returns the average of the numbers passed through the array as follows:

```
double getAverage(int arr[], int size)
{
```

```
int i;
double avg;
double sum;
for (i = 0; i < size; ++i)
{
sum += arr[i];
}
avg = sum / size;
return avg;
  }
```

Now, let us call the function:

```
#include<stdio.h>
/* function declaration */
double getAverage(int arr[], int size);
int main ()
{
/* an int array with 5 elements */
int balance[5] = {1000, 2, 3, 17, 50};
double avg;
/* pass pointer to the array as an argument */
avg = getAverage( balance, 5 ) ;
/* output the returned value */
printf( "Average value is: %f ", avg );
return 0;
}
```

When the code is compiled together and executed, it produces the following result: Average value is: 214.400000

5.6 Processing the Arrays

For certain applications the assignment of initial values to elements of an array is required. This means that the array be defined globally (extern) or locally as a static array.

Example:

Write a program to display the average marks of each student, given the marks in 2 subjects for 3 students.

```
/* Program to display the average marks of 3 students */
# include < stdio.h >
# define SIZE 3
main()
{
int i = 0;
float stud_marks1[SIZE];      /* subject 1array declaration */
float stud_marks2[SIZE];      /*subject 2 array declaration */
```

```
float total_marks[SIZE];
float avg[SIZE];
printf("\n Enter the marks in subject-1 out of 50 marks: \n");
for( i = 0;i<size;i++)
{
printf("Student no. =%d",i+1);
printf(" Please enter the marks= ");
scanf("%f",&stud_marks2[i]);
}
for(i=0;i<size;i++)
{
total_marks[i]=stud_marks1[i]+ stud_marks2[i];
avg[i]=total_marks[i]/2;
printf("Student no.=%d, Average= %f\n",i+1, avg[i]);
}
}
```

OUTPUT:

Enter the marks in subject-1out of 50 marks:
Student no. = 1 Enter the marks= 23
Student no. = 2 Enter the marks= 35
Student no. = 3 Enter the marks= 42

Enter the marks in subject-2 out of 50 marks:
Student no. = 1 Enter the marks= 31
Student no. = 2 Enter the marks= 35
Student no. = 3 Enter the marks= 40

Student no. = 1 Average= 27.000000
Student no. = 2 Average= 35.000000
Student no. = 3 Average= 41.000000

5.7 Multi-Dimensional Arrays

C programming language allows multidimensional arrays. Here is the general form of a multidimensional array declaration:

type name[size1][size2]...[sizeN];

Example:

The following declaration creates a three-dimensional integer array:

int three dim[5][10][4];

Now you are writing a chess-playing program. A chessboard is an 8-by-8 grid. What data structure would you use to represent it? You could use an array that has a chessboard-like structure, i.e. a two-dimensional array, to store the positions of the chess pieces. Two-dimensional arrays use two indices to pinpoint an individual element of the array. This is very similar to what is called "algebraic notation", commonly used in chess circles to record games and chess problems. I

In principle, there is no limit to the number of subscripts (or dimensions) an array can have. Arrays with more than one dimension are called multi- dimensional arrays. While humans cannot easily visualize objects with more than three dimensions, representing multi-dimensional arrays presents no problem to computers.

Multi-Dimensional Arrays Declaration

To declare an array of two dimensions as follows:

datatype array_name[size1][size2];

In the example of, variable_type is the name of some type of variable, such as int. Also, size1 and size2 are the sizes of the array's first and second dimensions, respectively.

Example:

Remember, because C arrays are zero-based, the indices on each side of the chessboard array run 0 through 7, rather than 1 through 8. The effect is the same: a two-dimensional array of 64 elements. int chessboard [8][8];

Initialization Two-Dimensional Arrays

The simplest form of multidimensional array is the two-dimensional array. A two-dimensional array is, in essence, a list of one-dimensional arrays.

To declare a two-dimensional integer array of size [x][y], To write something as:

type arrayName [x][y];

An m x n array, it will have m * n elements and will require m*n*elementsize bytes of storage. To allocate storage for an array you must reserve this amount of memory. The elements of a two-dimensional array are stored row wise.

```
int table [ 2 ] [ 3 ] = { 1,2,3,4,5,6 };
```

It means that element

```
table [ 0][0] = 1;
table [ 0][1] = 2;
table [ 0][2] = 3;
table [ 1][0] = 4;
table [ 1][1] = 5;
table [ 1][2] = 6;
```

The neutral order in which the initial values are assigned can be altered by including the groups in { } inside main enclosing brackets, the initialization as:

```
int table [ 2 ] [ 3 ] = { {1,2,3},
                          {4,5,6} };
```

The value within innermost braces will be assigned to those array elements whose last subscript changes most rapidly. If there are few remaining values in the row, they will be assigned zeros. The number of values cannot exceed the defined row size.

```
int table [ 2 ] [ 3 ] = { { 1, 2, 3},{ 4}};
```

It assigns as

```
table [0][0] = 1;
table [0][1] = 2;
table [0][2] = 3;
table [1][0] = 4;
table [1][1] = 0;
table [1][2] = 0
```

5.8 Array of Strings

Array of strings are multiple strings, stored in the form of table. Declaring array of strings is same as strings, except it will have additional dimension to store the number of strings.

An array of strings is a data structure that stores a collection of strings in contiguous memory locations, where each element of the array holds a single string. Unlike arrays of integers or other primitive data types, an array of strings allows for the storage and manipulation of sequences of characters, making it particularly useful for handling text-based data.

Declaration and Initialization: In languages like C, an array of strings can be declared and initialized as follows:

```
char *strings[] = {"string1", "string2", "string3"};
```

Accessing Elements: Individual strings within the array can be accessed using array indexing. For example:

```
char *firstString = strings[0];
```

Iterating Through the Array: You can iterate through the array of strings using loops like for or while. For example:

```
for (int i = 0; i < size; i++) {
   printf("%s\n", strings[i]);
```

Manipulating Strings: Since each element of the array is a string, you can perform string manipulation operations such as concatenation, comparison, and substring extraction on individual strings within the array.

Dynamic Allocation: In languages like C, you can dynamically allocate memory for an array of strings using pointers and memory allocation functions like malloc or calloc. This allows for flexibility in storing strings of varying lengths.

```
char **strings = malloc(n * sizeof(char *));
for (int i = 0; i < n; i++) {
   strings[i] = malloc(MAX_STRING_LENGTH * sizeof(char));
}
```

Memory Management: When working with dynamically allocated arrays of strings, it's important to free the allocated memory after use to avoid memory leaks.

```
// C
for (int i = 0; i < n; i++) {
```

```
    free(strings[i]);
}
free(strings);
```

5.9 Conclusion

The group has a single name for all its members, with the individual member being selected by an index. We have learnt in this unit, the basic purpose of using an array in the program, declaration of array and assigning values to the arrays. All elements of the arrays are stored in the consecutive memory locations. Without exception, all arrays in C are indexed from 0 up to one less than the bound given in the declaration.

One important point about array declarations is that they don't permit the use of varying subscripts. The numbers given must be constant expressions which can be evaluated at compile time, not run time. As with other variables, global and static array elements are initialized to 0 by default, and automatic array elements are filled with garbage values. In C, an array of type char is used to represent a character string, the end of which is marked by a byte set to 0 (also known as a NULL character).

5.10 Questions

1. **What is an array in C and how is it declared?**

Ans.1 An array in C is a collection of elements of the same data type stored in contiguous memory locations. It is declared by specifying the data type of its elements, followed by the array name and the size of the array in square brackets. For example, to declare an array of 10 integers, you would write:

```
int myArray[10];
```

2. **How do you initialize an array in C at the time of declaration?**

Ans.2 An array can be initialized at the time of declaration by providing a comma-separated list of values enclosed in curly braces. For example:

```
int myArray[5] = {1, 2, 3, 4, 5};
```

If the array size is omitted, the array will be sized automatically based on the number of values provided:

```
int myArray[] = {1, 2, 3, 4, 5};
```

3. **How can you access and modify elements in an array?**

Ans.3 Elements in an array can be accessed and modified using their index, with the first element having an index of 0. For example, to access the third element of an array **myArray**, you would use **myArray[2]**. To modify this element, you can assign a new value to it:

```
myArray[2] = 10;
```

4. **What happens if you try to access an array element out of its bounds in C?**

Ans.4 Accessing an array element out of its bounds (i.e., using an index that is negative or greater than or equal to the array size) leads to undefined behaviour. This can cause your program to crash or produce incorrect results, as you may be accessing or modifying memory that is not allocated for the array.

5. **Explain the concept of a multidimensional array in C with an example.**

Ans.5 A multidimensional array in C is an array of arrays. For example, a 2D array can be thought of as a matrix. It is declared by specifying the number of rows and columns:

```
int matrix [3][4];
```

You can initialize it similarly:

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

Access elements using two indices, one for the row and one for the column:

```
int value = matrix [1][2]; // value is 7
```

6. How can you copy the contents of one array to another in C?

Ans.6 You can copy the contents of one array to another using a loop to iterate through each element and assign it to the corresponding element in the target array. For example:

```
int source Array [5] = {1, 2, 3, 4, 5};
int destination Array [5];
for (int i = 0; i < 5; i++)
{
   destination Array[i] = source Array[i];
}
```

5.11 References

- Kernighan, B. W., & Ritchie, D. M. (1988). The C Programming Language (2nd ed.). Prentice Hall.
- King, K. N. (2008). C Programming: A Modern Approach (2nd ed.). W. W. Norton & Company.
- Prata, S. (2014). C Primer Plus (6th ed.). Addison-Wesley Professional.
- Deitel, P., & Deitel, H. (2016). C How to Program (8th ed.). Pearson.

***