C Programming with Arrays

Dear friends, welcome to the topic arrays in C programming. I hope that you have the concept of algorithms, programming languages especially the character set, basic syntax which includes the identifiers, reserved words or key words etc.

The previous lectures already covered the concept of data types and introduced the basic data types **int**, **char**, **float** and **double.** Please recall that the basic data types can be qualified by using data type qualifiers like short, long, signed and unsigned. I hope that you know the difference between a constant and a variable. A constant is a quantity that remains unchanged during the execution of a program where as a variable is an identifier that can take different values and represents a memory location of the data storage.

Now come to the concept of data type. Data type is a term that refers to the kind of data used in a program. Every programming language has a set of built-in data types. C supports several data types mainly under two major categories. The first category is ***Scalar data types*** (Fundamental data types) and **Derived data types** or structured data types.

As you know scalar data type is used for representing single value only. Since the data items of scalar data types are mere numbers and hence they are also known as arithmetic data types. The four scalar data types int, char, float and double are known as primitive or primary/fundamental/basic data types.

Now let us understand what is mean by derived data types. The data types derived from scalar data types with additional relationships between their elements are referred as derived data type or structured data types. C supports the derived data types including **Arrays, Functions, Pointers**, **Structures** and **Unions**.

So far, we have used only fundamental data types in writing C programs. However, in certain situations it will be easier to write programs using derived data types. Array is an important derived data structure in C programming language.

In many occasions we may want to process data of same type. For example, sorting of a list of 100 numbers in ascending or descending order. In this situation we have to declare 100 variables say a1, a2, a3 ..., a100 to store the entire list of numbers to carrying out the *sort* process. Now we can realise that the declaration of 100 variables to hold values corresponding to the elements present in the list is not a good idea. The adaptive way to cope up this objective is to use the concept of arrays that uses a common name with a subscript representing each element.

An array is an ordered sequence of finite data items of the same data type that shares a common name. The common name is the array name and each individual data item is known as an element of the array. The ordered sequence of such finite data items could be percentage marks of 100 students, number of chairs in your home, salaries of 300 employees or ages of 25 students. Thus an array is a collection of similar elements. These similar elements could be all integers or all floats or all characters etc. Usually, the array of characters is called a “string”, where as an array of integers or floats is called simply an array. The elements of the array are stored in the subsequent memory locations starting from the memory location given by the array name. It is important to remember that all elements of any given array must be of the same type i.e. we can’t have an array of 10 numbers, of which 5 are integers and 5 are floats.

We can use arrays to represent not only simple list of values but also table of data in two or three or more dimensions. That is an array may be one-dimensional or multidimensional. A one-dimensional array can be used to represent a list of data items. It is also known as a **vector.** A two-dimensional array can be used to represent a table of data items consisting of rows and columns. It is known as matrix. A three dimensional array can be used to represent a collection of tables and the concept can also be extended to arrays with more than three-dimensions also.

The major objectives of this lecture are to

- Differentiate between array and subscripted variables

- Declaration of one-dimensional and multidimensional arrays
- Explain the storage of array elements
- Initialise, read and display numeric arrays, character arrays
- Manipulate the elements of the arrays

Now let us move to the declaration of arrays.

Declaration of Arrays

Every array must be declared before we use it like other variables. The declarations of one-dimensional and multidimensional arrays differ slightly. Now we will see the syntax for the declaration of arrays along with the order of its storage with suitable examples.

I 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. In C, a single-subscripted variable xi (here x is the variable name and i the subscript) can be represented as,

x[0], x[1], x[2],.....,x[n-1].

Note:

In C, subscript can begin with number 0.

(a) Declaration of One-dimensional arrays:

As I told earlier, like any other variable, arrays must be declared before they are used. The syntax of declaring one-dimensional array is,

data_type var_name_1[size_1], var_name_2[size_2]....., var_name_n[size_n];
where

data_type : refers to the data type of elements in the array. It can be a primitive or derived data type

var_name_x :is an identifier which represents the array name.

size_x : is an expression representing the total number of elements in the array

for example,

int num[20];

It declares the **num** to be an array containing 20 integer elements.

Here any subscripts 0 to 19 are valid.

Note:

The subscripts of an array can be integer constants, integer variables, or expressions that yield integers. A one-dimensional array uses a single subscript enclosed within square brackets.

For example, num[10], num[i], num[4+5], num[i++]

C performs no bound checking and, therefore, care should be exercised to ensure that the array indices are within the declared limits.

For example,

For the declaration, int num[5] ; the access to the array element num[6] or num[5] won't create any error.

In this case **num** can only refer num[0], num[1],...., num[4]

The base address of an array is the address of the zero th element (starting element) of that array. When an array is declared, the compiler allocates a base address and reserves enough space in memory for all the elements of the array. In C the array name represents this base address.

For example, float x[5] is the declaration of a one-dimensional array. It defines a float array x of size 5 that represents a block of 5 consecutive storage regions. Here each element in the array is referred by the array variables x[0], x[1], x[2], x[3] and x[4] where 0,1,2,3 and 4 represent **subscripts** or **indices** of the array. In general, x[i]

refers to the ith element of the array. An array variable is also known as subscripted variable.

It is important to note that the array subscripts always start at 0 in C and they are integer expressions. Hence, x[0] refers to the starting element, x[1] refers to the next element and x[4] refers to the last element.

The declaration,

double **list[10], array[15];**

declares two arrays, named list and array having 10 and 15 elements respectively of double precision data type. The identifier array is also a valid array name and it is not a key word.

The declaration,

Char str[20];

declares character array **str** having a maximum of 20 characters.

(b) Initialization of One-dimensional arrays

The elements of an array may be assigned with values using initialization instead of reading them by the I/O functions. An array can be initialised in its declaration only. The general form of initialization of one dimensional array is

data_type variable_name[size] = { list of values };

The list of values is enclosed in braces, separated by commas and they must be constants or constant expressions.

For example int num[5] = { 10, 11, 12, 13, 14 };

where num[0] is initialised with 10, num[1] with 11, ….., num[4] with 14. The values within the braces are scanned from left and assigned to num[0], num[1] and so on. A semicolon should be placed after the closing brace.

Flavours of initialization:

(i) size may be omitted

int arr[ ] = { 1, 2, 3 }; will declare the arr array to contain three elements initialised with 1,2 and 3.

(ii) Initialization of Array elements to 0

int sum[5] = { 0 };

all elements in sum array is initialised to 0.

(iii) Implicit initialization

int arr[5] = { 1, 2 };

The first two elements in arr array is initialised with 1 and 2 and the remaining elements are initialised with 0.

(c) Reading and writing of one-dimensional arrays

As we already discussed, an array is a single entity holding a sequence of data items of similar data type. All types of arrays except character array can not be read/write as a single entity, but the individual array elements can be read/write.
For example, the elements of array arr are read and write (displayed) as

```
int arr[3];
scanf("%d%d%d", &arr[0], &arr[1], &arr[2]);
printf("%d%\nd%\nd\n", a[0],a[1],a[2]);
```

when the array size is large, then it is very tedious to read and display as we discussed. Here the loop structure may be used so as to simplify the task.

For example

```
for(i=0; i<3; i++)
        {
        scanf("%d", &arr[i]);
        printf("%d\n", arr[i]);
        }
```

It may be noted that within the loop, the variable i is initialised as 0 to start the subscript at zero. After reading the array, the elements can be accessed in nay order

and can be used in the same way as scalar variables.

Exercise 1

Write a program to read an array and print it in the reverse order.

```
/* Program for reversing an Array*/
#define MAX 5
main()
{
int i,x[MAX];
printf("Enter the array elements \n");
for (i=0;i<MAX;i++)
scanf("%d", &x[i] );  /* Reading array elements */
printf("Array elements in reverse order\n");
for(i=MAX-1;i>=0;i--)
printf("%d\t", x[i]);  /* Reversing the array */
}
```

II Two-dimensional Arrays

We have already seen that a two-dimensional array can be used to represent a table of data items consisting of rows and columns known as matrix. Now we will see how can we declare, initialise and read/write two dimensional arrays.

(a) Declaration of Two-dimensional arrays

The general syntax for declaring two dimensional array is

data_type var_name_1[row_size1] [col_size1], var_name_2[row_size2] [col_size2] , ..., var_name_n[row_sizen] [col_sizen];

Here two subscripts in two pairs of square brackets are used. The first subscript represents the row size and the second subscript represents the column size. The value in the ith row and jth column is referred by var_name[i][j].

For example, int **mat**[3][2]; declares a two dimensional integer array or we can say a 3x2 matrix named **mat**. The six elements of **mat** are stored row by row in the consecutive memory locations as shown in figure 1. This method of storing the elements in the memory is known as **row major order** storage representation.

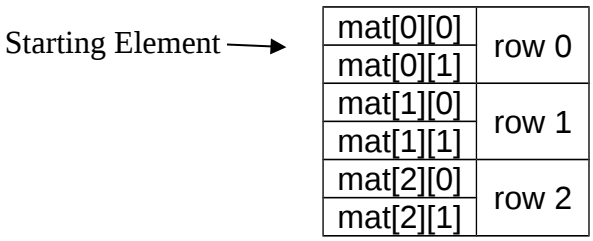


Figure 1. Storage representation of a two dimensional array

The zero Last Element **nat** is denoted by mat[0][0] and the last element is represented by mat[2][1].

Note:

Each dimension of the array is indexed from zero to its maximum size minus one; the first index selects the row and second index selects the column within the row.

The total number of elements in a two dimensional array is calculated by multiplying the number of rows by the number of columns.

(b) Initialization of Two-dimensional arrays

Like the one-dimensional arrays, two-dimensional arrays may be initialized by following their declaration with a list of initial values enclosed in braces.

For example, int mat[3][2] = { 1, 2, 3, 4, 5, 6};

Here the initialised values are assigned to the array elements (row by row) according to the order of the storage in the memory. The above statement can be equivalently written as,

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

This initialization enforces the assignment of each row, that is values within the inner braces are assigned to each row.

Flavours of initialization:

(i) Row size can be omitted in general in the case of multidimensional arrays the

leftmost subscript may be omitted and all others must be specified.

int mat[ ][3] = { {1, 2, 3}, {4, 5, 6});

(ii) The unspecified elements initialised automatically with zero

int mat[2][3] = { {1},{4,5}}; where first row elements are initialised with 1,0 and 0 and second row elements are initialised with 4, 5 and 0.

(iii) If the number of initialises exceeds the size of the array, it gives an error.

(c) Reading and writing of Two-dimensional arrays

Two dimensional arrays can be easily read and displayed using nested loop structures. We have explain this with the help of a C program written for reading a matrix and display the same and its transpose in kthe matrix format.

Let us go through a C program to find the transpose of a matrix

```
# define MAXROW 2
# define MAXCOL 4
Main()
{
int mat[MAXROW][MAXCOL], trans[MAXCOL][MAXROW];
int i,j;
printf("Enter the matrix values\n");
for(i=0;i<MAXROW;i++)
for(j=0;j<MAXCOL;j++)
        scanf("%d",&mat[i][j]);
for(i=0;i<MAXCOL;i++)
for(j=0;j<MAXROW;j++)
        trans[i][j] = mat[j][i];
printf("\n Transpose of the matrix \n\n");
for(i=0;i<MAXCOL;i++)
    {
        for(j=0;j<MAXROW;j++)
        printf("%d\t", trans[i][j]);
        printf("\n");
```

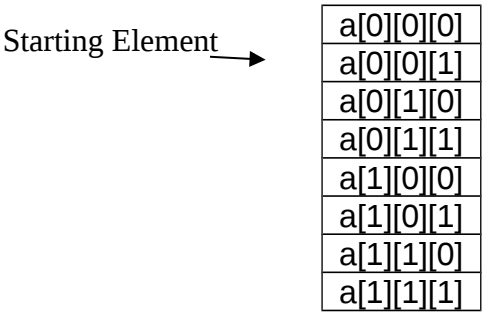
```
    }
}
```

III Multidimensional Arrays

C allows arrays of three or more dimensions. The maximum number of dimensions supported is compiler dependent. The general form of a multidimensional array is,

data_type var_name_1[size1] [size2]...[sizen], var_name_2[size1] [size2]... [sizen], ..., var_name_n[size1] [size2]...[sizen];

It is very important to understand the way in which the elements in a multidimensional array are stored. This will help to access the array elements. In the case of three dimensional array int a[2][2][2]; the compiler reserves the memory locations in the order given below.



IV Character arrays

A string is a one dimensional array of characters terminated by a NUL character. It Last Element e entity, unlike other type of arrays.

Character arrays can be initialised as

char test[] = {'a','e','i','o','u','\0'}; or

char test[] = "aeiou"; NUL character is automatically upended in this case.

Now let us see an example indicating how a tow-dimensional character array is initialised.

Char winter[3][10] = {“OCTOBER”, “NOVEMBER”, “DECEMBER”};

Net we will go through the library functions used for reading and printing string as a whole are given below.

String input /output functions

Reading from the keyboard	Writing to the screen	Remark
scanf() with %s as conversion specification	printf()	scanf() reads a string until a white space is encountered
gets()	puts()	gets() reads a string until a new line is encountered

When a string is used to read using the string input/output functions the NUL character is ‘\0’ is automatically inserted at the end of the string. Hence the size of the array must be equal to the number of characters plus 1. Even though NUL is not a part of normal text, it is included to mark the end of the string.

When puts() is used the name of the character array displays the string given as its argument. The delimiter character ‘\0’ of the argument string is converted into a new line character ‘\n’ when the string is displayed. The name of the array is the only argument for puts(). Whenever an error occurs, puts() returns EOF; otherwise, a non negative value is returned.

The function scanf() can be used to read a string by using the conversion specification %s. It reads a string until a white space character is read and the NUL is automatically upended to the string. The function printf() using %s as the conversion specification displays the string.

There are several other ways to read a string. The characters are read one by one using getchar(), or scanf() with the conversion specification %c, within a loop structure. The NUL character is not appended automatically using these functions and hence it is necessary to add it in a separate statement. Care must be taken to include the NUL character while reading a string using these functions to mark the end of the string. Also the size of the array must be chosen to include all characters including NUL.

Let us now see a C program which read a character array in different ways.

```
Main()
{
char temp1[20], temp2[20];
/* reading a string using scanf() with %s */
printf(“Enter a string and press any white space character \n”);
scanf(“%s”, temp1);
printf(“\n Entered string 1 is :\n“);
printf(“%s\n”,temp1);
/* reading a string using gets() */
printf(“Enter a string and press the enter key \n”);
gets(temp2);
printf(“\n Entered string 2 is :\n“);
puts(temp2);
}
```

Points to remember

1. Array declaration reserves the memory during compilation for its elements and hence the size of array must be given in the declaration without fail.
2. Size of array must be an integer greater than 0. After declaration, the values of subscripted variables cannot be assumed to have any particular value.
3. Only in the declarations, the subscript mention the size and in other places they refer to the indices only. The subscript must be an integer and always start at zero.
4. In a multidimensional array, each individual element is denoted by enclosing each subscript within a separate pair of square brackets.
5. It is a good practice to mention the size using named constants, which facilitate easier program modifications.
6. It is illegal to refer to the elements that are out of the array bound. But C has no checks on the bounds of an array and hence the compiler will not produce any syntax error for it, but may produce unexpected results.

Summary

- An array is a derived data type. It holds one or more elements of the same data type and hence useful to store more data.
- An array name followed by the subscript (starts from zero) enclosed within the square braces refers to the individual element in it.
- Array elements are stored in consecutive memory locations. The starting memory location is represented by the array name and is known as the base address of the array.
- Subscripted array variables (uses subscript enclosed within square brackets) can be treated as ordinary variables.
- By increasing the value of the subscript by one, the data stored in the subsequent memory location is obtained.
- Accessing an array element is easier irrespective of the size of the array. The subscript gives the position of the element in the array.
- Arrays may be one-dimensional or multidimensional.

Please try to completely solve this **assignment** consists of 5 theory questions and 5 programs.

ASSIGNMENT

1. Define an array. What is a subscripted variable?
2. What is the allowed data type for the subscript?
3. How are one-dimensional and multidimensional arrays declared in a C program?
4. Write a C statement that defines an array called many_birds that holds 50 types of birds.
5. How is a string initialised in C?
6. Write a C program to find the minimum value in an array and its position in the array.
7. Write a C program to find the sum of any given n numbers.
8. Write a C program to search a given number in an array.
9. Write a C program to find the sum of squares of elements on the diagonal of a square matrix.
10. Write a C program to read a string and find the frequency of each character in that string.

Here is some books for your **Reference**

REFERENCE

1. B. W. Kernighan and D. M. Ritchie, The C Programming Language, Prentice-Hall, Englewood Cliffs, New Jersey, 1978.
2. Yashavant Kanetkar; Let us C, BPB Publications, New Delhi.
3. Greg Perry, Absolute Beginners' guide to C, SAMS publishing, April 1994.

Hope you start enjoying the lessons of C and interested in writing C programs. There are many more other features in C are remaining to explore. So let us wait for the coming sessions and till then bye.