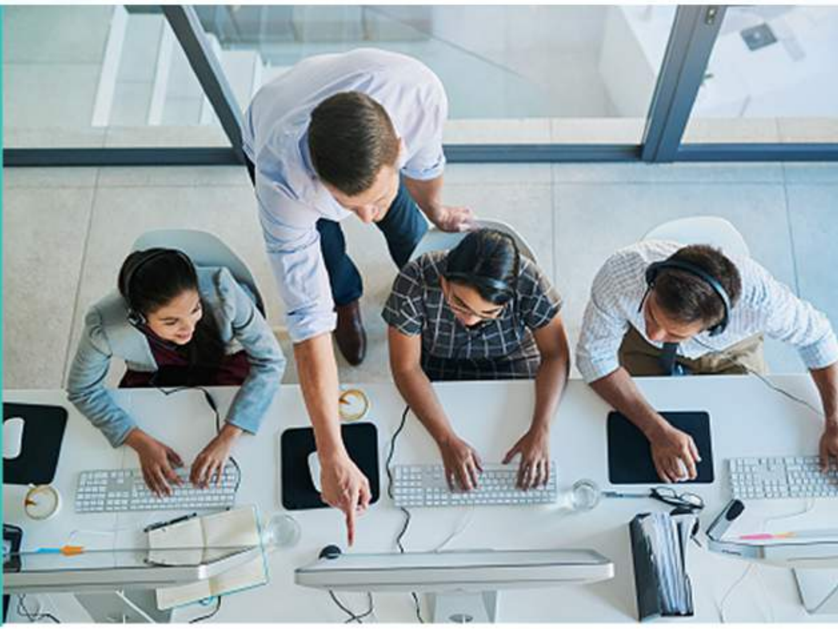
Andhra Pradesh State Skill Development Corporation

Skill AP
APSSDC

e Skill AP
Learn Anytime Anywhere

Andhra Pradesh State Skill Development Corporation



Programming in C
Arrays

ARRAYS

Array:

An array is a group (or collection) of the same data types. For example an int array holds the elements of int types while a float array holds the elements of float types.

Use of Array:

Consider a scenario where you need to find out the average of 100 integer numbers entered by the user. In C, you have two ways to do this:

1. Define 100 variables with int data type and then perform 100 scanf() operations to store the entered values in the variables and then at last calculate the average of them.
2. Have a single integer array to store all the values, loop the array to store all the entered values in the array and later calculate the average.
3. Obviously the second solution, it is convenient to store the same data types in one single variable and later access them using array index.

Declaration of Array in C:-

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 −

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. For example, to declare a 10-element array called balance of type double, use this statement −

double balance[10];

Here balance is a variable array which is sufficient to hold up to 10 double numbers.

```
int num[35]; /* An integer array of 35 elements */
char ch[10]; /* An array of characters for 10 elements */
```

Similarly an array can be of any data type such as double, float, short etc.

Initializing Arrays:-

You can initialize an array in C either one by one or using a single statement as follows:

```
double balance[5] = {1000.0, 2.0, 3.4, 7.0, 50.0};
```

The number of values between braces { } cannot be larger than the number of elements that we declare for the array between square brackets [ ].

If you omit the size of the array, an array just big enough to hold the initialization is created. Therefore, if you write −

```
            double balance[] = {1000.0, 2.0, 3.4, 7.0, 50.0};
```

You will create exactly the same array as you did in the previous example. Following is an example to assign a single element of the array −

```
                                        balance[4] = 50.0;
```

The above statement assigns the 5th element in the array with a value of 50.0. All arrays have 0 as the index of their first element which is also called the base index and the last index of an array will be the total size of the array minus 1. Shown below is the pictorial representation of the array we discussed above −

Accessing Arrays:-

You can use array subscript (or index) to access any element stored in the array. Subscript starts with 0, which means arr[0] represents the first element in the array arr.

In general arr[n-1] can be used to access the nth element of an array. where n is any integer number.

example:

```
int mydata[20];
mydata[0] /* first element of array mydata*/
mydata[19] /* last (20th) element of array mydata*/
```

Example of Array In C programming to find out the average of 4 integers

```
#include <stdio.h>
int main()
{
int avg = 0;
int sum =0;
int x=0;
/* Array- declaration – length 4*/
int num[4];
/* We are using a for loop to traverse through the array
* while storing the entered values in the array
*/
for (x=0; x<4;x++)
{
printf("Enter number %d \n", (x+1));
scanf("%d", &num[x]);
}
for (x=0; x<4;x++)
{
sum = sum+num[x];
}
avg = sum/4;
printf("Average of entered number is: %d", avg);
return 0;
}
```

Output:

```
Enter number 1
10
Enter number 2
10
Enter number 3
20
Enter number 4
40
Average of entered number is: 20
```

Important points in Program:

Input data into the array;-

Here we are iterating the array from 0 to 3 because the size of the array is 4. Inside the loop we are displaying a message to the user to enter the values. All the input values are stored in the corresponding array elements using the scanf function.

```
for (x=0; x<4;x++)
{
        printf("Enter number %d \n", (x+1));
        scanf("%d", &num[x]);
}
```

Reading out data from an array

Suppose, if we want to display the elements of the array then we can use the for loop in C like this.

```
for (x=0; x<4;x++)
{
        printf("num[%d]\n", num[x]);
}
```

Various ways to initialize an array:-

In the above example, we have just declared the array and later we initialized it with the values input by the user. However you can also initialize the array during declaration like this:

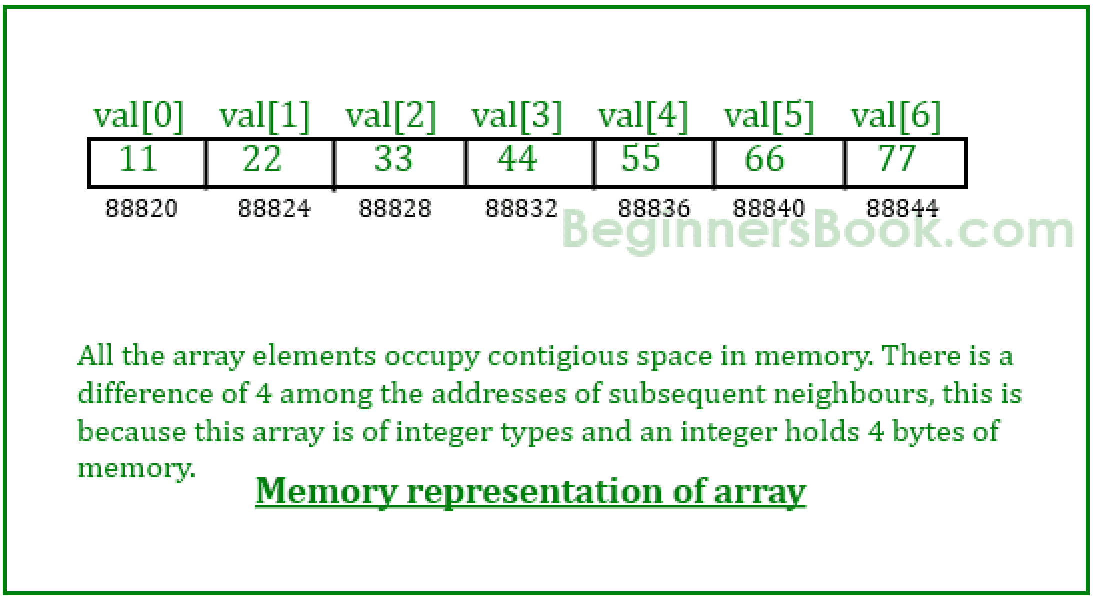
```
int arr[5] = {1, 2, 3, 4 ,5};
```

OR (both are same)

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

Un-initialized arrays always contain garbage values.

C Array – Memory representation:-



All the array elements occupy contigious space in memory. There is a difference of 4 among the addresses of subsequent neighbours, this is because this array is of integer types and an integer holds 4 bytes of memory.

Memory representation of array

Multidimensional Arrays:-

Multidimensional arrays can be described as "arrays of arrays". For example, a bidimensional array can be imagined as a two-dimensional table made of elements, all of them of the same uniform data type.

We can have multidimensional arrays in C like 2D and 3D arrays. However the most popular and frequently used array is 2D – two dimensional array. In this post you will learn how to declare, read and write data in 2D array along with various other features of it.

Two dimensional (2D) arrays in C programming with example:-

An array of arrays is known as 2D array. The two dimensional (2D) array in C programming is also known as matrix. A matrix can be represented as a table of rows and columns. Before we discuss more about two Dimensional arrays let's have a look at the following C program.

Simple Two dimensional(2D) Array Example:-

For now don’t worry how to initialize a two dimensional array, we will discuss that part later. This program demonstrates how to store the elements entered by the user in a 2d array and how to display the elements of a two dimensional array.

```
#include<stdio.h>
int main(){
/* 2D array declaration*/
        int disp[2][3];
        /*Counter variables for the loop*/
        int i, j;
        for(i=0; i<2; i++) {
                for(j=0;j<3;j++) {
                        printf("Enter value for disp[%d][%d]:", i, j);
                        scanf("%d", &disp[i][j]);
                }
        }
        //Displaying array elements
        printf("Two Dimensional array elements:\n");
        for(i=0; i<2; i++) {
                for(j=0;j<3;j++) {
                        printf("%d ", disp[i][j]);
                        if(j==2){
                                printf("\n");
                        }
                }
        }
        return 0;
}
```

Output:

```
Enter value for disp[0][0]:1
Enter value for disp[0][1]:2
Enter value for disp[0][2]:3
Enter value for disp[1][0]:4
Enter value for disp[1][1]:5
Enter value for disp[1][2]:6
Two Dimensional array elements:
1 2 3
4 5 6
```

Initialization of 2D Array:-

There are two ways to initialize two Dimensional arrays during declaration.

```
int disp[2][4] = { {10, 11, 12, 13}, {14, 15, 16, 17}};
OR
int disp[2][4] = { 10, 11, 12, 13, 14, 15, 16, 17};
```

Although both the above declarations are valid, I recommend you to use the first method as it is more readable, because you can visualize the rows and columns of the 2d array in this method.

Things that you must consider while initializing a 2D array:-

We already know, when we initialize a normal array (or you can say one dimensional array) during declaration, we need not to specify the size of it. However that’s not the case with 2D arrays, you must always specify the second dimension even if you are specifying elements during the declaration. Let's understand this with the help of few examples –

```
/* Valid declaration*/
int abc[2][2] = {1, 2, 3 ,4 }
/* Valid declaration*/
int abc[][2] = {1, 2, 3 ,4 }
/* Invalid declaration – you must specify second dimension*/
int abc[][] = {1, 2, 3 ,4 }
/* Invalid because of the same reason mentioned above*/
int abc[2][] = {1, 2, 3 ,4 }
```

How to store user input data into 2D array;-

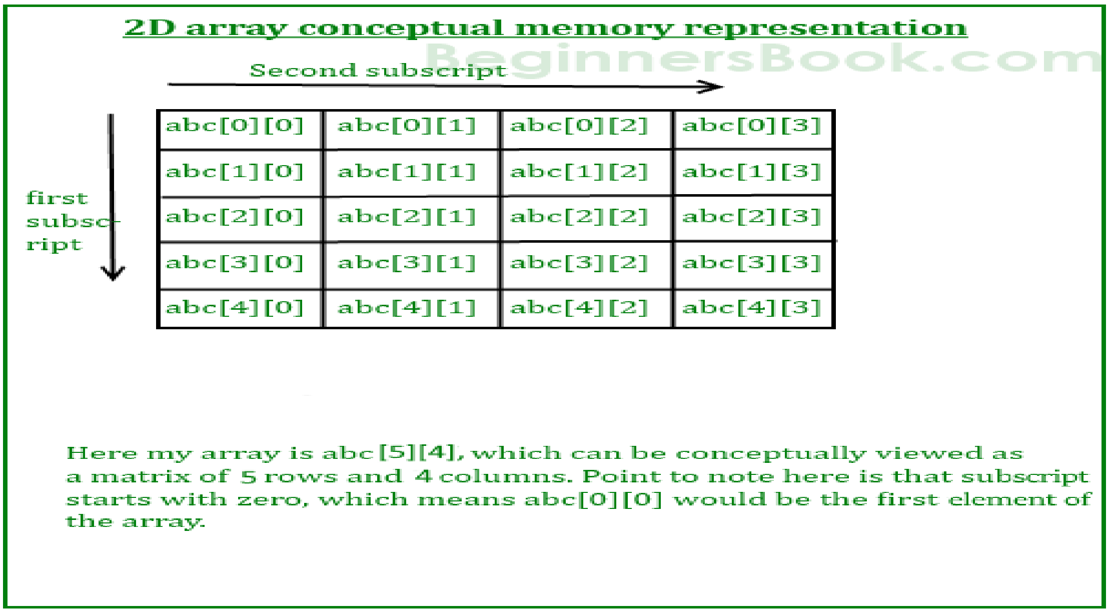
We can calculate how many elements a two dimensional array can have by using this formula: The array arr[n1][n2] can have n1*n2 elements. The array that we have in the example below is having the dimensions 5 and 4. These dimensions are known as subscripts. So this array has first subscript value as 5 and second subscript value as 4.
So the array abc[5][4] can have 5*4 = 20 elements.

To store the elements entered by the user we are using two for loops, one of them is a nested loop. The outer loop runs from 0 to the (first subscript -1) and the inner for loops runs from 0 to the (second subscript -1). This way the order in which the user enters the elements would be abc[0][0], abc[0][1], abc[0][2]…so on.

```
#include<stdio.h>
int main(){
        /* 2D array declaration*/
        int abc[5][4];
        /*Counter variables for the loop*/
        int i, j;
        for(i=0; i<5; i++) {
                for(j=0;j<4;j++) {
                        printf("Enter value for abc[%d][%d]:", i, j);
                        scanf("%d", &abc[i][j]);
                }
        }
        return 0;
}
```

In the above example, I have a 2D array abc of integer type. Conceptually you can visualize the above array like this:

Image:



2D array conceptual memory representation

Second subscript →

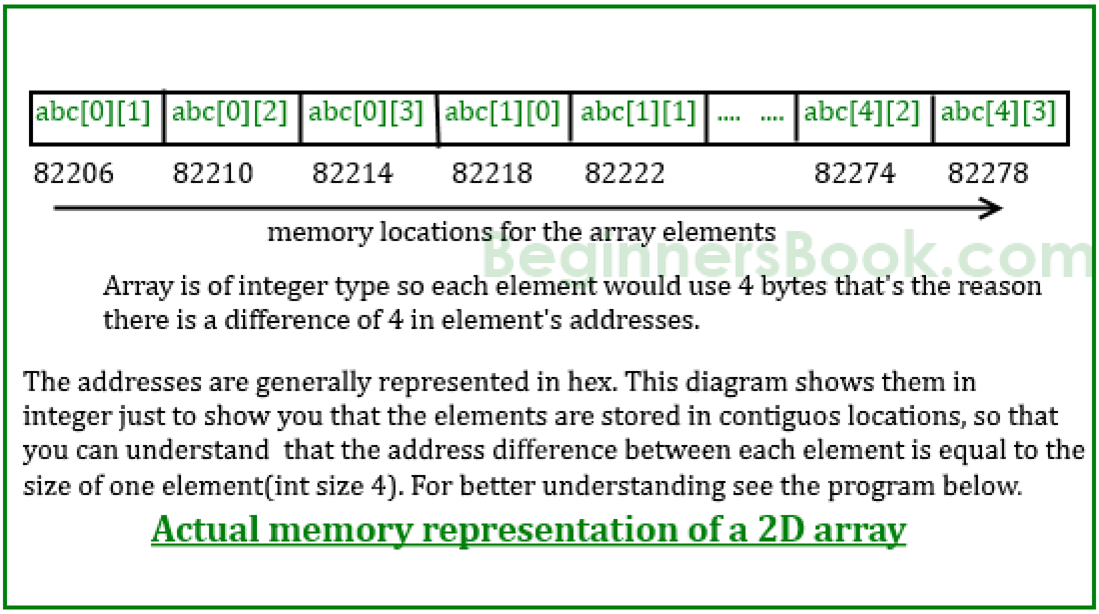
first subscript ↓

abc[0][0]	abc[0][1]	abc[0][2]	abc[0][3]
abc[1][0]	abc[1][1]	abc[1][2]	abc[1][3]
abc[2][0]	abc[2][1]	abc[2][2]	abc[2][3]
abc[3][0]	abc[3][1]	abc[3][2]	abc[3][3]
abc[4][0]	abc[4][1]	abc[4][2]	abc[4][3]

Here my array is abc[5][4], which can be conceptually viewed as a matrix of 5 rows and 4 columns. Point to note here is that subscript starts with zero, which means abc[0][0] would be the first element of the array.

However the actual representation of this array in memory would be something like this:

Image:



abc[0][1]	abc[0][2]	abc[0][3]	abc[1][0]	abc[1][1]	....	abc[4][2]	abc[4][3]
82206	82210	82214	82218	82222		82274	82278

memory locations for the array elements →

Array is of integer type so each element would use 4 bytes that's the reason there is a difference of 4 in element's addresses.

The addresses are generally represented in hex. This diagram shows them in integer just to show you that the elements are stored in contiguos locations, so that you can understand that the address difference between each element is equal to the size of one element(int size 4). For better understanding see the program below.

Actual memory representation of a 2D array

Passing array to function in C programming with example:-

Just like variables, arrays can also be passed to a function as an argument . In this guide, we will learn how to pass the array to a function using call by value and call by reference methods. But first we need to learn about

1)Function call by value in C

2)Function call by reference in C

Function call by value in C programming:-

Function call by value is the default way of calling a function in C programming. Before we discuss function call by value, lets understand the terminologies that we will use while explaining this:

Actual parameters: The parameters that appear in function calls.

Formal parameters: The parameters that appear in function declarations.

For example:

```
#include <stdio.h>
int sum(int a, int b)
{
        int c=a+b;
        return c;
}
int main(
{
        int var1 =10;
        int var2 = 20;
        int var3 = sum(var1, var2);
        printf("%d", var3);
        return 0;
}
```

In the above example variable a and b are the formal parameters (or formal arguments). Variable var1 and var2 are the actual arguments (or actual parameters). The actual parameters can also be the values. Like sum(10, 20), here 10 and 20 are actual parameters.

In this guide, we will discuss function call by value. If you want to read call by reference method then refer to this guide: function call by reference.

What is Function Call By value?

When we pass the actual parameters while calling a function then this is known as function call by value. In this case the values of actual parameters are copied to the formal parameters. Thus operations performed on the formal parameters don’t reflect in the actual parameters.

Example of Function call by Value

As mentioned above, in the call by value the actual arguments are copied to the formal arguments, hence any operation performed by function on arguments doesn’t affect actual parameters. Let's take an example to understand this:

```
#include <stdio.h>
```

```
int increment(int var)
{
        var = var+1;
        return var;
}
int main()
{
        int num1=20;
        int num2 = increment(num1);
        printf("num1 value is: %d", num1);
        printf("\nnum2 value is: %d", num2);
        return 0;
}
```

Output:

```
num1 value is: 20
num2 value is: 21
```

Explanation:

We passed the variable num1 while calling the method, but since we are calling the function using call by value method, only the value of num1 is copied to the formal parameter var. Thus change made to the var doesn’t reflect in the num1.

Example 2: Swapping numbers using Function Call by Value

```
#include <stdio.h>
void swapnum( int var1, int var2 )
{
        int tempnum ;
        /*Copying var1 value into temporary variable */
        tempnum = var1 ;
        /* Copying var2 value into var1*/
        var1 = var2 ;
        /*Copying temporary variable value into var2 */
        var2 = tempnum ;
}
int main( )
{
        int num1 = 35, num2 = 45 ;
        printf("Before swapping: %d, %d", num1, num2);
        /*calling swap function*/
        swapnum(num1, num2);
        printf("\nAfter swapping: %d, %d", num1, num2);
}
```

Output:

```
Before swapping: 35, 45
After swapping: 35, 45
```

Why do variables remain unchanged even after the swap?

The reason is the same – function is called by value for num1 & num2. So actually var1 and var2 gets swapped (not num1 & num2). As in call by value actual parameters are just copied into the formal parameters.

Function call by reference :-

Before we discuss function call by reference, lets understand the terminologies that we will use while explaining this:

Actual parameters: The parameters that appear in function calls.
Formal parameters: The parameters that appear in function declaration.

For example: We have a function declaration like this:

```
int sum(int a, int b);
```

The a and b parameters are formal parameters.

We are calling the function like this:

```
int s = sum(10, 20); //Here 10 and 20 are actual parameters
or
int s = sum(n1, n2); //Here n1 and n2 are actual parameters
```

In this guide, we will discuss function call by reference method. If you want to read call by value method then refer to this guide: function call by value.

What is Function Call By Reference?

When we call a function by passing the addresses of actual parameters then this way of calling the function is known as call by reference. In call by reference, the operation performed on formal parameters, affects the value of actual parameters because all the operations performed on the value stored in the address of actual parameters. It may sound confusing first but the following example would clear your doubts.

Example of Function call by Reference;-

Let's take a simple example. Read the comments in the following program.

```
#include <stdio.h>
void increment(int *var)
{
        /* Although we are performing the increment on variable
        * var, however the var is a pointer that holds the address
        * of variable num, which means the increment is actually done
        * on the address where value of num is stored.
        */
        *var = *var+1;
}
int main()
{
        int num=20;
        /* This way of calling the function is known as call by
        * reference. Instead of passing the variable num, we are
        * passing the address of variable num
        */
        increment(&num);
        printf("Value of num is: %d", num);
        return 0;
}
```

Output:

```
Value of num is: 21
```

Example 2: Function Call by Reference – Swapping numbers:-

Here we are swapping the numbers using call by reference. As you can see the values of the variables have been changed after calling the swapnum() function because the swap happened on the addresses of the variables num1 and num2.

```
#include
void swapnum ( int *var1, int *var2 )
{
        int tempnum ;
        tempnum = *var1 ;
        *var1 = *var2 ;
        *var2 = tempnum ;
}


int main( )
{
        int num1 = 35, num2 = 45 ;
        printf("Before swapping:");
        printf("\nnum1 value is %d", num1);
        printf("\nnum2 value is %d", num2);
        /*calling swap function*/
        swapnum( &num1, &num2 );
        printf("\nAfter swapping:");
        printf("\nnum1 value is %d", num1);
        printf("\nnum2 value is %d", num2);
return 0;
}
```

Output:

```
Before swapping:
num1 value is 35
num2 value is 45
After swapping:
num1 value is 45
num2 value is 35
```

Passing array to function using call by value method:-

As we already know in this type of function call, the actual parameter is copied to the formal parameters.

```
#include <stdio.h>
void disp( char ch)
{
        printf("%c ", ch);
}
int main()
{
        char arr[] = {'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j'};
        for (int x=0; x<10; x++)
{
/* I'm passing each element one by one using subscript*/
disp (arr[x]);
}
return 0;
}
```

Output:-

```
a b c d e f g h i j
```

Passing array to function using call by reference:-

When we pass the address of an array while calling a function then this is called function call by reference. When we pass an address as an argument, the function declaration should have a pointer as a parameter to receive the passed address.

```
#include <stdio.h>
void disp( int *num)
{
printf("%d ", *num);
}
int main()
{
int arr[] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 0};
for (int i=0; i<10; i++)
{
/* Passing addresses of array elements*/
disp (&arr[i]);
}
return 0;
}
```

Output:-

```
1 2 3 4 5 6 7 8 9 0
```

How to pass an entire array to a function as an argument?

In the above example, we have passed the address of each array element one by one using a for loop in C. However you can also pass an entire array to a function like this:

Note:-

The array name itself is the address of the first element of that array. For example if array name is arr then you can say that arr is equivalent to the &arr[0].

```
#include <stdio.h>
void myfuncn( int *var1, int var2)
{
        /* The pointer var1 is pointing to the first element of
        * the array and the var2 is the size of the array. In the
        * loop we are incrementing pointer so that it points to
        * the next element of the array on each increment.
        *
        */
        for(int x=0; x<var2; x++)
        {
                printf("Value of var_arr[%d] is: %d \n", x, *var1);
                /*increment pointer for next element fetch*/
                var1++;
        }
}
int main()
{
        int var_arr[] = {11, 22, 33, 44, 55, 66, 77};
        myfuncn(var_arr, 7);
        return 0;
}
```

Output:-

```
Value of var_arr[0] is: 11 Value of var_arr[1] is: 22 Value of var_arr[2] is: 33 Value of var_arr[3]
is: 44 Value of var_arr[4] is: 55 Value of var_arr[5] is: 66 Value of var_arr[6] is: 77.
```