**1D, 2D, MultiDimensional Array Assignments**

Mandatory

1D Array

1. Refer the code snippet and answer the queries

int main()

{

int array[100];

int \*ptr;

// do something

}

Q1: Can pointer be used in Array-style syntax? e.g. ptr[10], ptr[0]

**Yes, pointer can be used in array-style syntax. In fact, this is one of the valid ways to access elements in a pointer. ptr[10] is equivalent to \*(ptr + 10) and ptr[0] is equivalent to \*(ptr + 0) or simply \*ptr.**

**Since ptr is a pointer to an int, if ptr is properly initialized to point to an array or a block of memory, ptr[10] will give you the 11th element in the array or memory block (due to zero-based indexing).**

Q2: Can Array be used in Pointer-style syntax? e.g. \*array, \*(array + 0), \*(array + 10)

**Yes, an array can be used in pointer-style syntax. In C, the name of an array (array) is implicitly converted to a pointer to the first element of the array. Therefore, \*array is equivalent to array[0], \*(array + 0) is also equivalent to array[0], and \*(array + 10) is equivalent to array[10].**

Q3: is ptr++ valid?

**Yes, ptr++ is valid. The ++ operator increments the pointer to the next memory location based on the type of pointer. Since ptr is a pointer to int, ptr++ will increment ptr to point to the next int in memory (i.e., it moves the pointer by sizeof(int) bytes).**

**If ptr is initially pointing to array[0], after ptr++, it will point to array[1]**

Q4: is array++ valid?

**No, array++ is not valid. In C, the name of an array is a constant pointer to the first element, and it cannot be incremented or modified. So, the expression array++ will cause a compilation error because array is not a modifiable lvalue. The pointer ptr, on the other hand, can be incremented, but array cannot.**

Q5: what is sizeof(array)?

**sizeof(array) will give the total size of the entire array, not just a pointer. Since array is an array of 100 int elements, and assuming int has a size of 4 bytes, the result will be:**

**sizeof(array) = 100 \* sizeof(int) = 100 \* 4 = 400 bytes**

**So, sizeof(array) will return 400 if int is 4 bytes.**

Q6: what is sizeof(ptr)?

**sizeof(ptr) gives the size of the pointer itself, not the memory it points to. A pointer typically takes up 4 or 8 bytes on most modern systems, depending on whether the system is 32-bit or 64-bit.**

* **On a 32-bit system, sizeof(ptr) will be 4 bytes.**
* **On a 64-bit system, sizeof(ptr) will be 8 bytes.**

**Therefore, sizeof(ptr) will return either 4 or 8 depending on the system architecture.**

2. Refer the code snippet below. Comment on the other elements (other than those that are explicitly initialized) of all array variables in code snippet below.

#define MAX 100

int main()

{

int arr[MAX] = {11,22,33};

int arr1[MAX]={0};

static int arr2[MAX];

}

**Array arr:**

int arr[MAX] = {11, 22, 33};

* **Explicitly initialized elements**: arr[0] = 11, arr[1] = 22, and arr[2] = 33.
* **Other elements**: All the other elements of the array (from arr[3] to arr[99]) will **automatically be initialized to 0**. This is because in C, when an array is partially initialized, all elements that are not explicitly initialized will be set to 0 by default.

**Array arr1:**

int arr1[MAX] = {0};

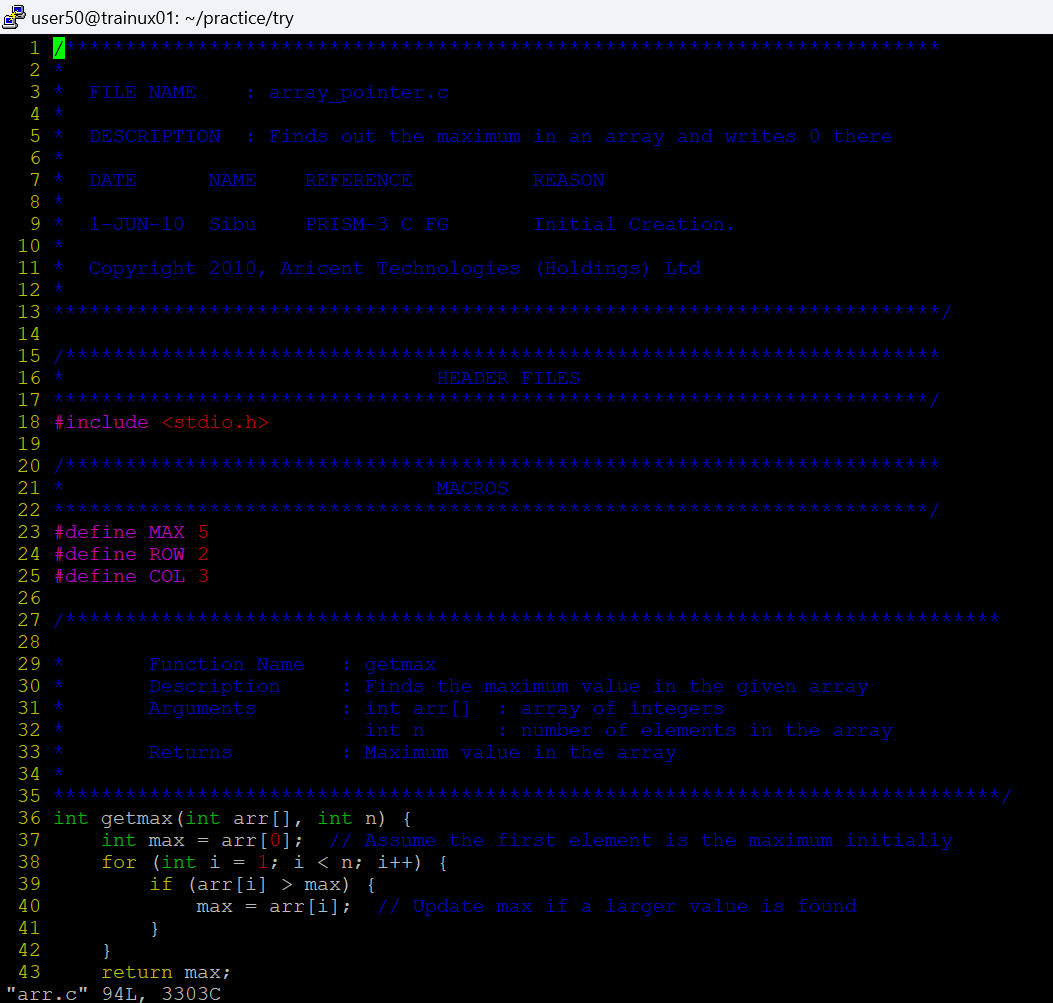
* **Explicitly initialized element**: arr1[0] = 0.
* **Other elements**: Since only arr1[0] is initialized and the rest of the array is implicitly initialized, **all elements of arr1 will be initialized to 0**. This is because in C, when an array is initialized with a single value, all elements of the array will be set to that value.

**Array arr2 (static):**

static int arr2[MAX];

* **Static storage duration**: The array arr2 is declared with the static keyword, meaning that it has static storage duration, and its lifetime is for the duration of the program.
* **Default initialization**: In C, arrays with static storage duration (such as arr2) are automatically initialized to 0 if no explicit initialization is provided. Therefore, all elements of arr2 will be initialized to 0.

3. Refer the program “array\_pointer.c”. Add a function getmax() to find the maximum in the array and call in main() and display the result.



A screenshot of a computer program

Description automatically generated

A computer screen with white text

Description automatically generated

A computer screen with white text

Description automatically generated

4. Extend the code given below to read N and a start value from the user to perform the given operations.

#define MAX 100

int main()

{

int arr[MAX] = {11,22,33};

}

Add the following functions choosing proper input, output and return.

a. init() - Use the inputs to initialize the first N elements of the array with N consequetive values starting with given start value .

b. update() – increment value of every element in the array

c. display() – display the contents of array

2D, MultiDimensional Arrays

1. Implement sort() to sort a given array. Refer the code snippet below.

int main()

{

char arr[]= “xaybz”;

sort(arr, sizeof(arr)/sizeof(arr[0]);

return 0;

}

2. Refer the code snippet below.

int main()

{

char arr[][3] = {

sort(arr, sizeof(arr)/sizeof(arr[0]);

return 0;

}

Allow user to perform the following operations.

a. init() - initialize the array and return 0

b. search\_update() – search for a given element in array and if found update it to given value and return 0 else return 1

c. display() – traverse and display array contents

For the functions, pass array and other required arguments to functions and return as per requirement



A screen shot of a computer program

Description automatically generated

A screenshot of a computer program

Description automatically generated

A screenshot of a computer screen

Description automatically generated