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, a pointer can be used in an array-style syntax like ptr[10] or ptr[0]. In fact, ptr[10] is equivalent to *(ptr + 10), and ptr[0] is equivalent to *ptr.

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

Yes, an array can be used in a pointer-style syntax. For example, *array is equivalent to array[0], and *(array + 0) is also equivalent to array[0]. Moreover, *(array + 10) accesses the 11th element of the array (index 10).

Q3: is ptr++ valid?

Yes, ptr++ is valid. It increments the pointer ptr to point to the next element in the array, assuming ptr is properly initialized and points to an array or a block of memory.

Q4: is array++ valid?

No, array++ is **not valid**. The name of an array is a constant pointer to its first element, and you cannot modify the address of the array itself. Therefore, you cannot increment the array name like a pointer.

Q5: what is sizeof(array)?

sizeof(array) gives the total size of the array in bytes. For example, if array is defined as int array[100];, then sizeof(array) will give the size of 100 * sizeof(int) (assuming sizeof(int) = 4, it will be 400 bytes).

Q6: what is sizeof(ptr)?

sizeof(ptr) gives the size of the pointer variable ptr. It is typically the size of a pointer on your system (4 bytes on 32-bit systems, 8 bytes on 64-bit systems), regardless of the type the pointer points to.

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];
}
```

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.

```c
#include <stdio.h>

#define MAX 100

int getmax(int arr[], int n) {
    int max = arr[0];
    for (int i = 1; i < n; i++) {
        if (arr[i] > max) {
            max = arr[i];
        }
    }
    return max;
}

int main() {
    int arr[MAX] = {11, 22, 33, 99, 7};
    int n = 5;

    int max_value = getmax(arr, n);
    printf("The maximum value in the array is: %d\n", max_value);

    return 0;
}
```

```
user72@trainux01:~/splint$ vi array.c
user72@trainux01:~/splint$ gcc array.c
user72@trainux01:~/splint$ ./a.out
The maximum value in the array is: 99
```

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

user60@trainux01: ~/Batch17OCT2024_175/Assignments/Day04/1D_2D_MultiDimensional_Array_Assignments

```c
#include <stdio.h>

#define MAX 100

void init(int arr[], int n, int start_value) {
    for (int i = 0; i < n; i++) {
        arr[i] = start_value + i;
    }
}

void update(int arr[], int n) {
    for (int i = 0; i < n; i++) {
        arr[i]++;
    }
}

void display(int arr[], int n) {
    for (int i = 0; i < n; i++) {
        printf("%d ", arr[i]);
    }
    printf("\n");
}

int main() {
    int arr[MAX];
    int n, start_value;
    printf("Enter number of elements (N): ");
    scanf("%d", &n);
    printf("Enter the start value: ");
    scanf("%d", &start_value);
    init(arr, n, start_value);
    printf("Initialized array: ");
    display(arr, n);
    update(arr, n);
    printf("Updated array: ");
    display(arr, n);

    return 0;
}
```

```
user72@trainux01:~/splint$ vi fdc.c
user72@trainux01:~/splint$ gcc fdc.c
user72@trainux01:~/splint$ ./a.out
Enter number of elements (N): 4
Enter the start value: 2
Initialized array: 2 3 4 5
Updated array: 3 4 5 6
```

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;
}
```

user60@trainux01: ~/Batch17OCT2024_175/Assignments/Day04/1D_2D_MultiDimensional_Array_Assignments

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

void sort(char arr[], int n) {
    for (int i = 0; i < n - 1; i++) {
        for (int j = i + 1; j < n; j++) {
            if (arr[i] > arr[j]) {
                char temp = arr[i];
                arr[i] = arr[j];
                arr[j] = temp;
            }
        }
    }
}

int main() {
    char arr[] = "xaybz";
    int n = sizeof(arr) / sizeof(arr[0]) - 1;

    sort(arr, n);
    printf("Sorted string: %s\n", arr);

    return 0;
}

```

```
user72@trainux01:~/splint$ vi sdc.c
user72@trainux01:~/splint$ gcc sdc.c
user72@trainux01:~/splint$ ./a.out
Sorted string: abxyz
```

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

user60@trainux01: ~/Batch17OCT2024_175/Assignments/Day04/1D_2D_MultiDimensional_Array_Assignments

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

#define ROWS 3
#define COLS 3

void init(int arr[ROWS][COLS]) {
    for (int i = 0; i < ROWS; i++) {
        for (int j = 0; j < COLS; j++) {
            arr[i][j] = 0;
        }
    }
}

int search_update(int arr[ROWS][COLS], int old_val, int new_val) {
    for (int i = 0; i < ROWS; i++) {
        for (int j = 0; j < COLS; j++) {
            if (arr[i][j] == old_val) {
                arr[i][j] = new_val;
                return 0;
            }
        }
    }
    return 1;
}

void display(int arr[ROWS][COLS]) {
    for (int i = 0; i < ROWS; i++) {
        for (int j = 0; j < COLS; j++) {
            printf("%d ", arr[i][j]);
        }
        printf("\n");
    }
}

int main() {
    int arr[ROWS][COLS] = {{1, 2, 3}, {4, 5, 6}, {7, 8, 9}};

    printf("Initial array:\n");
    display(arr);

    int old_val, new_val;
    printf("Enter value to search: ");
    scanf("%d", &old_val);
    printf("Enter new value: ");
    scanf("%d", &new_val);

    if (search_update(arr, old_val, new_val) == 0) {
        printf("Array after update:\n");
        display(arr);
    } else {
        printf("Element not found\n");
    }

    return 0;
}
```

```
user72@trainux01:~/splint$ vi udc.c
user72@trainux01:~/splint$ gcc udc.c
user72@trainux01:~/splint$ ./a.out
Initial array:
1 2 3
4 5 6
7 8 9
Enter value to search: 5
Enter new value: 8
Array after update:
1 2 3
4 8 6
7 8 9
```