1D, 2D, MultiDimensional Array Assignments

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]

Sol: Yes, a pointer can be used in array-style syntax. This is because ptr[10] is essentially equivalent to \*(ptr + 10).

* ptr[0] is the same as \*(ptr + 0)
* ptr[10] is the same as \*(ptr + 10)

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

Sol: Yes, arrays can also be used in pointer-style syntax. In C, the name of the array is a pointer to its first element. Thus:

* \*array is equivalent to array[0]
* \*(array + 0) is the same as array[0]
* \*(array + 10) is equivalent to array[10]

Q3: is ptr++ valid?

Sol: Yes, ptr++ is valid. It increments the pointer ptr to point to the next element in memory. If ptr is pointing to an integer array, then ptr++ will make it point to the next integer (it increments by the size of an integer, typically 4 bytes).

Q4: is array++ valid?

Sol: No, array++ is not valid. The name of an array in C is a constant pointer to the first element of the array, and thus we cannot change its base address.

Q5: what is sizeof(array)?

Sol: sizeof(array) returns the total size of the array in bytes.

ex: For int array[100], assuming the size of int is 4 bytes, sizeof(array) will be 4 \* 100 = 400 bytes.

Q6: what is sizeof(ptr)?

Sol: sizeof(ptr) will return the size of the pointer itself, not the array it points to. On most systems, the size of a pointer is 4 bytes (on 32-bit systems) or 8 bytes (on 64-bit systems). So, sizeof(ptr) will typically be 4 or 8 bytes, depending on the 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];

}

Sol:

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

* The first three elements of the array are explicitly initialized to 11, 22, and 33.
* The remaining elements (from arr[3] to arr[99]) will be automatically initialized to 0 because in C, arrays with partial initialization are zero-initialized for the remaining elements.

2**.** int arr1[MAX] = {0};

* The first element of the array is explicitly initialized to 0.
* All other elements (from arr1[1] to arr1[99]) are also automatically initialized to 0 because this is the behavior of partial initialization in C. If the first element is 0, all other elements are set to 0 as well.

3. static int arr2[MAX];

* The array arr2 is static, meaning it is initialized to zero by default. Static variables in C, whether explicitly initialized or not, are initialized to zero. Thus, all elements in arr2 will be zero.

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.

#include <stdio.h>

int getmax(int arr[], int size) {

int max = arr[0];

for (int i = 1; i < size; i++) {

if (arr[i] > max) {

max = arr[i];

}

}

return max;

}

int main() {

int arr[] = {11, 22, 33, 44, 55};

int size = sizeof(arr) / sizeof(arr[0]);

int max\_value = getmax(arr, size);

printf("Maximum value: %d\n", max\_value);

return 0;

}

Output:

A number of numbers and symbols on a black background

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

sol :

#include <stdio.h>

#define MAX 100

int i;

// Function to initialize the array

void init(int arr[], int N, int start) {

for ( i = 0; i < N; i++) {

arr[i] = start + i; // Fill the array with consecutive values starting from 'start'

}

}

// Function to update the array by incrementing each element

void update(int arr[], int N) {

for (i = 0; i < N; i++) {

arr[i]++; // Increment the value at each position

}

}

// Function to display the array

void display(int arr[], int N) {

for (i = 0; i < N; i++) {

printf("%d ", arr[i]); // Print each element of the array

}

printf("\n");

}

int main() {

int arr[MAX] = {0}; // Initialize an array with 0s

int N, start;

// Get user inputs for N and the start value

printf("Enter the number of elements (N): ");

scanf("%d", &N);

if (N > MAX) {

printf("N should be less than or equal to %d.\n", MAX);

return 1;

}

printf("Enter the start value: ");

scanf("%d", &start);

// Initialize the array

init(arr, N, start);

// Display the array after initialization

printf("Array after initialization: ");

display(arr, N);

// Update the array

update(arr, N);

// Display the array after update

printf("Array after update: ");

display(arr, N);

return 0;

}

Output:

A screen shot of a computer

Description automatically generated

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;

}

Sol:

#include <stdio.h>

#include <string.h>

// Function to implement Sort for the array of characters

void sort(char arr[], int n) {

int i, j;

char temp;

// sorting

for (i = 0; i < n - 1; i++) {

for (j = 0; j < n - i - 1; j++) {

if (arr[j] > arr[j + 1]) {

// Swap elements if they are in the wrong order

temp = arr[j];

arr[j] = arr[j + 1];

arr[j + 1] = temp;

}

}

}

}

int main() {

char arr[] = "xaybz"; // Initialize the array of characters

int n = sizeof(arr) / sizeof(arr[0]) - 1; // Calculate the length of the array (excluding null terminator)

sort(arr, n); // Call the sort function to sort the array

// Output the sorted array

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

return 0;

}

Output:

A black background with white text

Description automatically generated

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

Sol:

#include <stdio.h>

#define ROWS 3

#define COLS 3

int i,j;

// Function to initialize the 2D array with a given character

int init(char arr[ROWS][COLS]) {

char values[ROWS][COLS] = {

{'a', 'b', 'c'},

{'d', 'e', 'f'},

{'g', 'h', 'i'}

};

for ( i = 0; i < ROWS; i++) {

for ( j = 0; j < COLS; j++) {

arr[i][j] = values[i][j]; // Copy the values into the array

}

}

return 0; // Return 0 to indicate successful initialization

}

// Function to search for an element in the 2D array and update it if found

int search\_update(char arr[ROWS][COLS], char old\_val, char new\_val) {

for ( i = 0; i < ROWS; i++) {

for ( j = 0; j < COLS; j++) {

if (arr[i][j] == old\_val) {

arr[i][j] = new\_val; // Update the value if found

return 0; // Return 0 indicating successful update

}

}

}

return 1; // Return 1 if the value is not found

}

// Function to display the contents of the 2D array

void display(char arr[ROWS][COLS]) {

for ( i = 0; i < ROWS; i++) {

for ( j = 0; j < COLS; j++) {

printf("%c ", arr[i][j]); // Print each character

}

printf("\n"); // New line after each row

}

}

int main() {

char arr[ROWS][COLS]; // Declare the 2D array

// Initialize the array

init(arr);

// Display the initialized array

printf("Array after initialization:\n");

display(arr);

// Example of searching and updating an element

char old\_val, new\_val;

printf("Enter the character to search for: ");

scanf(" %c", &old\_val);

printf("Enter the new character to update: ");

scanf(" %c", &new\_val);

// Search and update the element

if (search\_update(arr, old\_val, new\_val) == 0) {

printf("Array after updating '%c' to '%c':\n", old\_val, new\_val);

display(arr);

} else {

printf("Character '%c' not found in the array.\n", old\_val);

}

return 0;

}

Output:

