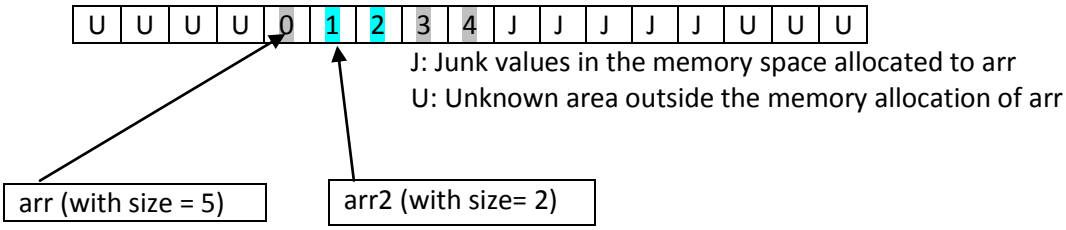
Arrays & Pointers: Basic Information and Simple Exercises

Capacity of an array is the memory allocated for the array in the name of variable it is defined. Size of an array indicates the amount of the used part of the array. It cannot be extracted from an array, so it is always stored in a variable and send to functions with the array. Examine the code segment below. Size of the array can never be understood in the function "sumArray" unless the size is sent to it, also, it can never be known in main unless it is returned by "fillArray".

```
#define CAPASITY 10 /*DO NOT use the word SIZE here*/
…
        int arr[CAPASITY], sizeArr, sumArr;
        int * arr2, sumArr2;
        fillArray(arr, &sizeArr);
        sumArr=sumArray(arr, sizeArr);
        arr2 = &arr[1]; /*Alternatively arr2=arr+1*/
        sumArr2=sumArray(arr2, 2) ;
….

void fillArray(int arr[], int* size){
        arr[0]=0; arr[1]=1; arr[2]; arr[3]=3 arr[4]=4
        *size=5;
}
int sumArray(int arr[], int size){
        int i, sum=0;
        for(i=0;  i<size; ++i)
                sum+=arr[i];
        return sum;
}
```

An array is a constant pointer showing its first element itself. It is considered as a contiguous block starting from where it shows and ending "size" variables after this point. Below a memory image for the above defined array is shown.



Be aware that any pointer with some size information can be used as an array. For example, arr2 defined in the following code segment has a memory image shown above and sumArr2 has the value of 3.

a. Write a function to print all elements of an integer array with the following prototype:

void printArray(int* array, int size)

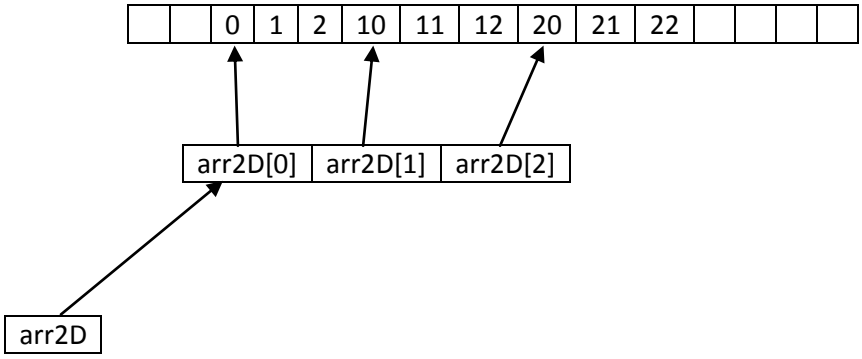
Define an integer array holding integers from 0 to 9. Send this array to "printArray" with the following size parameters: 15, 10, 5, 0. Explain what it prints.

Send the address of the 2nd and the 5th element of the array you defined to "printArray" so that it prints the array following from that element.

b. Recall the information below:

C also holds a multidimensional array as a contiguous memory block in one dimensional memory. For example, each line of a 2 dimensional array is stored one after another. Examine the array definition below and the memory image associated with it.

```
int arr2D[3][3]={{0, 1, 2}, {10, 11, 12}, {20, 21, 22}};
```



Note that:

- Memory is in fact one dimensional and a multidimensional array is stored contiguously in it, such that a part of a 2D array can easily be expressed by using a 1D array. Examine the code segment below:

```
int* arr1D = &arr2D[0][0];
int* arr1D2 = &arr2D[1][1];
int sum1, sum2;
sum1 = sumArray(arr1D, 5); /*Returns 0+1+2+10+11*/
sum2 = sumArray(arr1D2, 2); /*Returns 12+20*/
```

- A 2D array is an array of 1D arrays (and more generally an n dimensional array is an array of (n-1) dimensional arrays), such that each line of it can be reached directly using one dimensional array format. Examine the code segment below:

```
sumArray(arr2D[0], 2);   /*returns 0+1+2*/
sumArray(arr2D[1], 4); /*returns 10+11+12+20*/
```

Write a function accepting a 2D integer array having 10 columns and filing it with increasing numbers starting from 0 such that each line starts with the value one more than the last value of the previous line. Function will have the following prototype:

void fill2DArray(int array[][10], int numOfLines)

Obtain a 2D array with 5 lines using this function. Obtain some 1D arrays using integer pointers as described above from this array and send them to printArray function you wrote to print the shaded areas below:

0	1	2	3	4	5	6	7	8	9
10	11	12	13	14	15	16	17	18	19
20	21	22	23	24	25	26	27	28	29
30	31	32	33	34	35	36	37	38	39
40	41	42	43	44	45	46	47	48	49