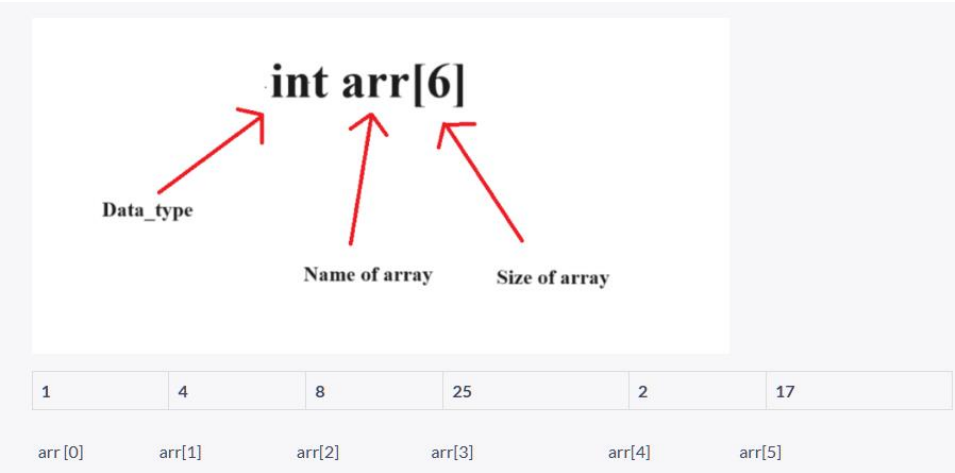
1. Define an array in C. What are its main characteristics and uses in programming?

An array is a collection of similar types of data items at a contiguous memory location. However, the size of the array is fixed, and you need to a lot its size at the time of declaration. to create an array, define the data type (like int)




```c
#include <stdio.h>

int main()
{
    int myNumbers[] = {1, 4, 8, 25, 2};
    printf("%d", myNumbers[3]);

    // Outputs 25
}
```

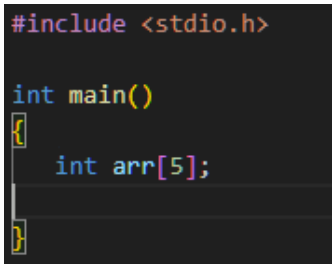
Why do we need Arrays?

Array stores an extensive collection of similar data types. We have only three variables, and then we can easily store them using separate names like int var1, int var2, and int var3, but what if we have an extensive collection of these variables? Then, we cannot assign separate names to each of them, as it would be tedious and time-consuming. Here comes the use of arrays in C, where we can store multiple data type values in a contiguous memory block. Hence, we can use an array in C when we are working with a large number of similar items.

Main Characteristics of Arrays in C:

Characteristic	Description
Same data type	**All elements in the array must be of the same type (e.g., all int or all float).**
Zero-based indexing	**The first element is at index 0, the second at 1, and so on.**
Contiguous memory	**Array elements are stored next to each other in memory.**
Fixed size	**Once declared, the size of the array cannot change during runtime.**
Fast access	**You can access any element quickly using its index (e.g., arr[2]).**

2. Explain how memory is allocated for a one-dimensional array in C. Why is it said that arrays are stored in contiguous memory?



When you declare a one-dimensional array in C like this:

You're telling the compiler to **allocate a block of memory**

that can hold 5 integers, each integer takes 4 Bytes

$$4 \times 5 = 20 \text{ bytes}$$

These 20 bytes are reserved in **one continuous block** in memory — no gaps in between.

Why??

Because when the array is stored in memory, each element is placed right after the previous one, in sequential (continuous) memory addresses, this makes it easy to access elements by index.

3. Describe how you can initialize an array in C. Give examples for full and partial initialization. What values do uninitialized elements take?

1. **Full Initialization:**

   You provide values for every element in the array.

```c
int main()
{
int arr[5] = {10, 20, 30, 40, 50};

}
```

2. **Partial Initialization:**

   You provide values for some elements only.

```c
int main()
{
int arr[5] = {10, 20};  // only first two elements initialized

}
```

Note: In partial initialization, any elements you don't specify are automatically set to 0 (zero) only if the array is initialized during declaration.

```c
#include <stdio.h>

int main()
{
   int arr[5];

}
```

This is **uninitialized**, the values of the elements are **garbage values**

4. How do you calculate the number of elements in a statically declared array in C? Why is this technique not applicable to dynamically allocated arrays?

How do you calculate the number of elements in a statically declared array in C:

```c
nt main()

   int arr[5] = {10, 20}; // only first two elements initialized
   int length = sizeof(arr) / sizeof(arr[0]);
   printf("%d\n", length);
   // output is 5
```

Why doesn’t it work with dynamic arrays:

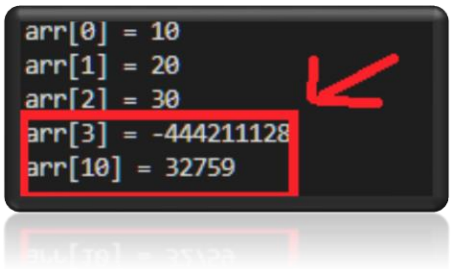
Because the compiler has no idea how much memory malloc allocated — it only sees a pointer.

We can create a dynamic array in C by using the following methods:

1. **Using malloc() Function**
2. **Using calloc() Function**
3. **Resizing Array Using realloc() Function**
4. **Using Variable Length Arrays(VLAs)**
5. **Using Flexible Array Members**

5.What are the consequences of accessing an array index outside its bounds? Why doesn’t the C compiler always detect this error?

Accessing an array index outside its valid range in C results in undefined behavior. This means the program may behave in unpredictable ways, depending on what memory is accessed. C gives the programmer direct access to memory without automatic safety checks, So, the programmer must be careful to stay within valid array limits.




```c
nt main()
{
    int arr[3] = {10, 20, 30};

    printf("arr[0] = %d\n", arr[0]);
    printf("arr[1] = %d\n", arr[1]);
    printf("arr[2] = %d\n", arr[2]);
    // Trying to reach elements outside size
    printf("arr[3] = %d\n", arr[3]);
    printf("arr[10] = %d\n", arr[10]);
    // give you  Random value OR undefined behavior.
}
```

6. Differentiate between a one-dimensional and a two-dimensional array with examples. What are typical use cases for each?

Difference Between One-Dimensional and Two-Dimensional Array

Parameters	One-Dimensional Array	Two-Dimensional Array
Basics	A one-dimensional array stores a single list of various elements having a similar data type.	A two-dimensional array stores an *array of various arrays*, or a *list of various lists*, or an *array of various one-dimensional arrays*.
Representation	It represents multiple data items in the form of a list.	It represents multiple data items in the form of a table that contains columns and rows.
Dimensions	It has only one dimension.	It has a total of two dimensions.
Parameters of Receiving	One can easily receive it in a pointer, an unsized array, or a sized array.	The parameters that receive it must define an array's rightmost dimension.
Total Size (in terms of Bytes)	Total number of Bytes = The size of array x the size of array variable or datatype.	Total number of Bytes = The size of array visible or datatype x the size of second index x the size of the first index.

Example of 1D Array:

```
int main()
{
    int numbers[5] = {10, 20, 30, 40, 50};

}
```

Example of 2D Array:

```
int main()
{
int matrix[2][3] = {
    {1, 2, 3},
    {4, 5, 6}
};
}
```

The array has 2 rows and 3 columns

Accessing element at row 1, column 2 would be matrix [1][2], which is 6

Elements are stored row by row in memory (row-major order)

How to Print a 2D Array: Using nested loops:

In C, a two-dimensional array is essentially an array of one-dimensional arrays.

7. How can you pass an array to a function in C? Explain what has passed and how this affects the original array.

In C, you can pass an array to a function by using its name only

What is passed to the function is not the entire array,

but only a pointer to the first element

```c
void printArray(int arr[], int size);

int main()
{
    int arr[3] = {1, 2, 3};
    printArray(arr, 3);
}
```

Does this affect the original array?

YES

Since the function works on the same memory (because of the pointer),
any change inside the function will affect the original array.

8. Describe the relationship between pointers and arrays in C. How can pointer arithmetic be used to access array elements?

In C, arrays and pointers are closely related.

When using pointer arithmetic with arrays in C, you're working directly with memory addresses instead of traditional indexing. Since the name of an array acts like a pointer to its first element, you can use arithmetic to move through the array by incrementing or offsetting the pointer

When you declare an array, the array name acts as a pointer to the first element of the array ex.

```c
int main()
{
    int arr[5] = {10, 20, 30, 40, 50};
    int *ptr = arr;
    printf("Using array notation: arr[2] = %d\n", arr[2]);
    printf("Using pointer notation: *(ptr + 2) = %d\n", *(ptr + 2));
    return 0;
}
```

9.What limitations do arrays have in C, and how can these be addressed using structures like dynamic arrays, linked lists, or vectors in other languages?

1. Fixed Size:
    - The size of an array must be known at compile time.
    - You can't change its size while the program is running.
2. No Bounds Checking:
    - C does not check if you access an element outside the array's range.
    - This may lead to undefined behavior or memory corruption.
3. Wasted or Insufficient Memory:
    - If you overestimate the size → you waste memory.
    - If you underestimate it → you might run out of space.
4. No Built-in Functions:
    - No automatic resizing, inserting, or deleting elements.
    - You must manage everything manually.
5. Hard to Insert/Delete Elements:
    - Inserting or deleting an element requires shifting others.
    - It's inefficient in terms of performance**.**

How to Address These Limitations:

1. Dynamic Arrays (Using malloc / realloc):

- Allows arrays to be resized during runtime.
- Allocated using malloc() and grown with realloc().
- Solves the fixed size problem.

2. Linked Lists:

- Memory is allocated as needed for each element.
- Easily insert/delete items without shifting.
- Slower for direct access, but flexible for growing.

3. Vectors (e.g., in C++):

- Dynamic arrays with built-in functions.
- Automatically resize as needed.
- Provide functions for adding/removing elements.
- Safer and easier to use than raw arrays.

10. What is the enum data type used for?

In C, an enumeration (or enum) is a user defined data type that contains a set of named integer constants. It is used to assign meaningful names to integer values, which makes a program easy to read and maintain.

Definition

An enum must be defined before we can use it in program.

```
enum enum_name {
    n1, n2, n3, ...
};
```

where, **n1, n2, n3, ...** are names assigned to an integer value. By default, first name **n1** is assigned **0, n2** is assigned **1** and the subsequent ones are **incremented by 1.**

Why use enum?

1. Gives meaningful names to integer values.
2. Improves code clarity.
3. Helps avoid using random numbers (magic numbers) in code.

Common Uses of enum:

1. **Representing fixed choices (like days, directions, states).**
2. **Improving code readability instead of using plain numbers.**
3. **Making decisions in if or switch statements.**
4. **Avoiding "magic numbers" in the code.**

```c
#include <stdio.h>
enum TrafficLight
{
    RED,
    YELLOW,
    GREEN
};

int main()
{
    enum TrafficLight light = RED;

    if (light == RED)
    {
        printf("Stop!");
    }
    else if (light == YELLOW)
    {
        printf("Wait...");
    }
    else if (light == GREEN)
    {
        printf("Go!");
    }
    return 0;
    // Output stop!
}
```

References:

1- https://www.w3schools.com/c/c_arrays.php
2- https://pwskills.com/blog/array-in-c/
3- https://www.simplilearn.com/tutorials/c-tutorial/array-in-c
4- https://www.geeksforgeeks.org/c/dynamic-array-in-c/
5- https://byjus.com/gate/difference-between-one-dimensional-and-two-dimensional-array/
6- https://www.naukri.com/code360/library/passing-arrays-to-functions-in-c-cpp
7- https://www.wscubetech.com/resources/c-programming/pointer-arithmetic
8- https://www.geeksforgeeks.org/c/enumeration-enum-c/