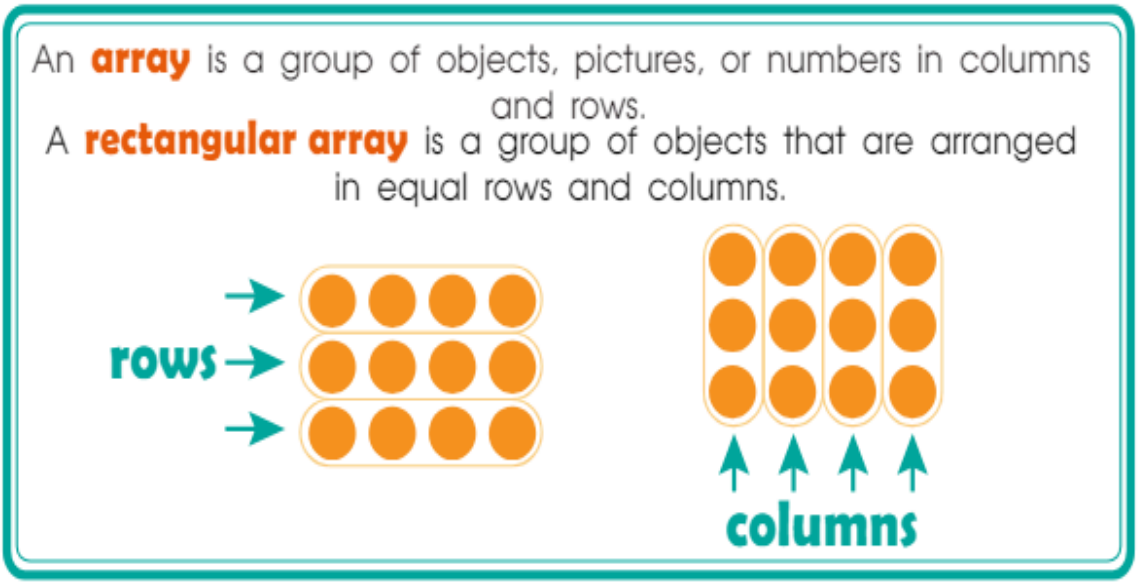
Day 9	Arrays

Objectives	Learn how to declare, initialize, and access elements in a one-dimensional array. Practice performing basic operations such as insertion, deletion, and traversal on one-dimensional arrays.
Outcomes	Students will be able to 1. Effectively use one-dimensional arrays to store and manage a collection of elements. 2.Write C code to perform basic operations on arrays, such as inserting and deleting elements.

Description

An array is a collection of one or more values of the same data type stored in contiguous memory locations. The data type can be user-defined or even any other primitive data-type. Elements of an array can be accessed with the same array name by specifying the index number as the location in memory.



In arrays we only store the similar elements in a container. A container is used to store multiple types of variables.

Types of Arrays

Arrays in C are classified into three types:

1. One-dimensional arrays
2. Two-dimensional arrays
3. Multi-dimensional arrays

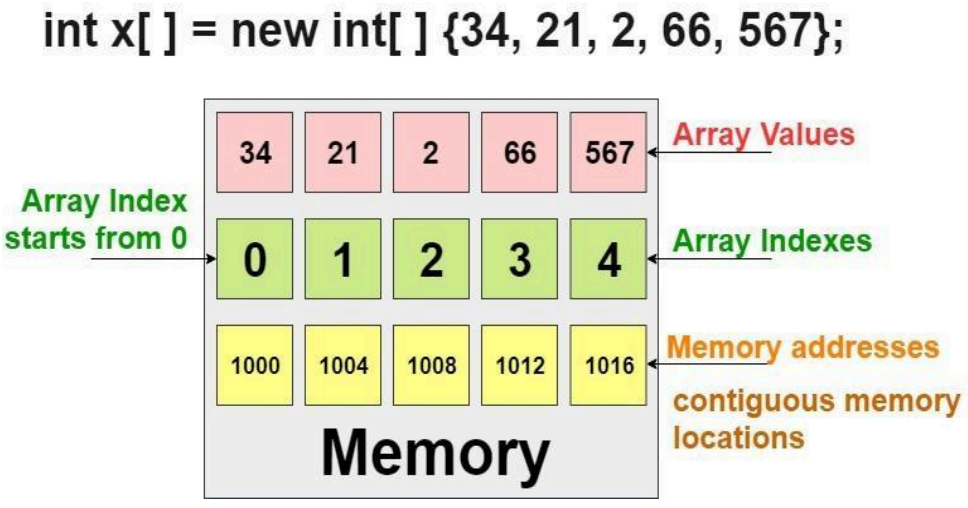
1. One Dimensional Array in C

Arrays are a fundamental concept in programming, and they come in different dimensions. One-dimensional arrays, also known as single arrays, are arrays with only one dimension or a single row. In this practice, we'll dive deep into one-dimensional arrays in C programming language, including their syntax, examples, and output.

Syntax

The syntax of a **one-dimensional array** in C programming language is as follows:

- **dataType** specifies the data type of the array. It can be any valid data type in C programming language, such as **int, float, char, double**, etc.
- **arrayName** is the name of the array, which is used to refer to the array in the program.
- **arraySize** specifies the number of elements in the array. It must be a **positive integer value** greater than zero, representing the number of elements that can be stored in the array.



Rules for Declaring One Dimensional Array in C

Declaring a one dimensional array in C involves a set of rules and regulations that must be strictly followed to ensure the correct functionality of the program. Here are some crucial rules:

Fixed Size

The size of the array should be defined at the time of declaration. The size must be a constant integer, i.e., it cannot be a variable or a dynamic expression.

```
int arr[5]; // Valid

int size = 5;

int arr[size]; // Invalid in C, size should be a constant
```

No Negative Dimension

The size of the array must be greater than zero. Negative or zero dimensions are not allowed.

```
int arr[5]; // Valid
int arr[0]; // Invalid
int arr[-5]; // Invalid
```

Data Type

The elements of an array should all be of the same data type. You cannot have an array with mixed data types.

```
int arr[5]; // Valid

int arr[5] = {1, "two", 3, "four", 5}; // Invalid, mixed data types
```

Array Initialization

If you initialize an array at the time of declaration, ensure that the number of elements does not exceed the specified size. You can, however, initialize an array with fewer elements than specified, and the remaining elements will automatically be assigned the default value of the data type (0 for integers, 0.0 for floats, '\0' for characters).

```
int arr[5] = {1, 2, 3, 4, 5}; // Valid

int arr[5] = {1, 2, 3}; // Valid, the remaining elements will be set to 0

int arr[5] = {1, 2, 3, 4, 5, 6}; // Invalid, more elements than specified
```

Array Name

The array name should follow the rules for identifiers in C. It can contain alphabets, digits, and underscores. However, it cannot begin with a digit.

```
int arr[5]; // Valid

int 1arr[5]; // Invalid, identifier starts with digit
```

Example

Let's take a simple example of a one-dimensional array in C programming language to understand its syntax and usage.

```
#include <stdio.h>

int main() {

int numbers[5] = {10, 20, 30, 40, 50};

for(int i=0; i<5; i++) {

printf("numbers[%d] = %d\n", i, numbers[i]);

}

return 0;
```

```
}
```

Output

```
numbers[0] = 10

numbers[1] = 20

numbers[2] = 30

numbers[3] = 40

numbers[4] = 50
```

Explanation

In the above example, we have declared a one-dimensional array of integers named **numbers**. The array contains five elements, and each element is initialized with a value.

We have used a **for loop** to iterate over the elements of the array and print their values using the **printf** function.

Initializing One Dimensional Array in C

In C programming language, we can initialize a one-dimensional array while declaring it or later in the program. We can initialize a one-dimensional array while declaring it by using the following syntax:

```
dataType arrayName[arraySize] = {element1, element2, ..., elementN};
```

- **"dataType'** specifies the data type of the array.

- **"arrayName'** is the name of the array.

- **"arraySize'** specifies the number of elements in the array.

- **"{element1, element2, ..., elementN}'** specifies the values of the elements in the array. The number of elements must be equal to **"arraySize"**.

There are two primary methods of initializing a one dimensional array in C.

a. Compile-Time Initialization

This method involves initialising an array at the time of its declaration. In this case, the size and elements of the array are defined at compile time.

Example

```
int main() {
  int num[5] = {1, 2, 3, 4, 5};
  return 0;
}
```

In the above example, the array 'num' is being initialized with 5 elements during compile-time.

b. Run-Time Initialization

In run-time initialization, the array is initialized during the execution of the program, not at the compile time. This is particularly useful when inserting values into an array based on user input or some computations.

Example

```
int main() {
  int num[5];
  for(int i=0; i<5; i++) {
    printf("Enter the element for index %d: ", i);
    scanf("%d", &num[i]);
  }
  return 0;
```

```
}
```

In this example, the program will ask the user to enter five integers, one at a time, which will be stored in the array 'num'. We can also initialize a ***one-dimensional array*** later in the program by assigning values to its elements using the following syntax:

```
arrayName[index] = value;
```

- ***arrayName*** is the name of the array.

- **index** is the index of the element we want to assign a value to.

- **value** is the value we want to assign to the element.

Accessing Elements of One-Dimensional Array in C

In a one-dimensional array, each element is identified by its index or position in the array. The index of the first element in the array is 0, and the index of the last element is arraySize - 1.

To access an element of a one-dimensional array in C programming language, we use the following

Syntax

```
arrayName[index]
```

- **arrayName** is the name of the array.
- **index** is the index of the element we want to access.

Example

Let's take an example to understand how to access elements of a one-dimensional array in C programming language.

```
#include <stdio.h>

int main() {

int numbers[5] = {10, 20, 30, 40, 50};

printf("The first element of the array is: %d\n", numbers[0]);

printf("The third element of the array is: %d\n", numbers[2]);
```

```
return 0;

}
```

Output

The first element of the array is: 10

The third element of the array is: 30

Explanation

In the above example, we have declared a one-dimensional array of integers named ***numbers***. We have accessed the first element of the array using the ***index 0*** and the third element of the array using the ***index 2***.

Modifying Elements of One-Dimensional Arrays

We can modify the value of individual elements of a one-dimensional array using their index. To modify an element, we simply assign a new value to it using the ***assignment operator*** =.

Example

```
#include <stdio.h>

 int main() {

  int numbers[5] = {10, 20, 30, 40, 50};

printf("The third element of the array is %d\n", numbers[2]);

numbers[2] = 35;

    printf("The third element of the array is now %d\n", numbers[2]);

     return 0;

 }
```

Output

The third element of the array is 30

The third element of the array is now 35

Explanation

In the above example, we have declared a one-dimensional array of integers named ***numbers*** and initialized it with the values ***{10, 20, 30, 40, 50}***. We have used the ***printf*** function to print the third element of the array, which is accessed using the ***index 2***.

After that, we modified the value of the ***third element*** by assigning a new value of ***35*** to it. Finally, we have used the ***printf*** function again to print the new value of the third element.

Copying One-Dimensional Arrays

To copy the contents of one array to another, we need to copy the individual elements.

Examples

```
int main() {
   int num1[5] = {1, 2, 3, 4, 5};
   int num2[5];
   for(int i=0; i<5; i++) {
      num2[i] = num1[i];
   }
   // Printing elements of array num2
   for(int i=0; i<5; i++) {
      printf("%d ", num2[i]);
  }
   return 0;
}
```

In the above example, the elements of 'num1' are copied to 'num2' using a for loop.

Guess the below snippet is Valid or not

Int array1[]={10,20,30,60.78}
Not valid
int array2[]={10,20,30,40,50}
Valid
char str[14] = { 'G','e','e','k','s','f','o','r','G','e','e','k','s',"bit"};
Not valid
char str[14] = { 'G','e','e','k','s','f','o','r','G','e','e','k','s','u'};
Valid

Note:

Array start index is 0 and Array End index is -1

Maximum and Minimum element in an array

```
#include <stdio.h>

int main()
{
   int arr1[100];
   int i, mx, mn, n;

   // Prompt user for input
   printf("\n\nFind maximum and minimum element in an array :\n");
   printf("-------------------------------------------------\n");
   printf("Input the number of elements to be stored in the array :");
   scanf("%d", &n);

   // Input elements for the array
   printf("Input %d elements in the array :\n", n);
   for (i = 0; i < n; i++)
   {
      printf("element - %d : ", i);
      scanf("%d", &arr1[i]);
   }
```

```
    // Initialize max (mx) and min (mn) with the first element of the array
    mx = arr1[0];
    mn = arr1[0];

    // Traverse the array to find the maximum and minimum elements
    for (i = 1; i < n; i++)
    {
        // Update mx if the current element is greater
        if (arr1[i] > mx)
        {
            mx = arr1[i];
        }

        // Update mn if the current element is smaller
        if (arr1[i] < mn)
        {
            mn = arr1[i];
        }
    }

    // Print the maximum and minimum elements
    printf("Maximum element is : %d\n", mx);
    printf("Minimum element is : %d\n\n", mn);
            return 0;
}
```