Arrays, String, Pointers & Reference

INDEX

Table of Contents

[Arrays in C++ 2](#_Toc143052792)

[Array declaration 2](#_Toc143052793)

[Multidimensional Array 3](#_Toc143052794)

[Memory Layout 4](#_Toc143052795)

[Strings in C++ 5](#_Toc143052796)

[C-style (character arrays and literals) 5](#_Toc143052797)

[String Class in C++ STL 6](#_Toc143052798)

[String manipulations 7](#_Toc143052799)

[Pointers & References in C++ 12](#_Toc143052800)

[Pointers vs References 12](#_Toc143052801)

[Pointers 13](#_Toc143052802)

[Use case 13](#_Toc143052803)

[Reference 16](#_Toc143052804)

[Use case 16](#_Toc143052805)

[Types of Function Call (References v/s Pointers) 17](#_Toc143052806)

[Call-by-Value 17](#_Toc143052807)

[Call-by-Reference with Pointer Arguments 18](#_Toc143052808)

[Call-by-Reference with Reference Arguments 19](#_Toc143052809)

[Array Name as Pointers 20](#_Toc143052810)

# Arrays in C++

Array is a collection of similar types of data elements, stored at continuous memory locations. It is a built-in data type.

A colorful rectangular object with numbers and symbols

Description automatically generated

## Array declaration

by specifying size: int arr[10]

by initializing elements: int arr[] = { 10, 20, 30, 40 }

## Multidimensional Array

A black background with white text

Description automatically generated

For example - A 2D array defined as follows can be viewed as a table of two rows and three columns.



An N-dimensional array is an array of arrays. It will be mapped to one-dimensional memory addresses.



## Memory Layout

A contiguous memory space is allocated for array elements and can be accessed via an array index.

A diagram of a number of numbers

Description automatically generated with medium confidence

# Strings in C++

There are mainly 2 ways of handling strings in C++.

1. **C-style** strings or character arrays.
2. string template class

## C-style (character arrays and literals)

A screenshot of a computer program

Description automatically generated

## String Class in C++ STL

A screen shot of a computer

Description automatically generated

### String manipulations

**Remember**

An **iterator** is an object (like a pointer) that points to an element inside the container. Containers are



Each container class in the C++ Standard Library provides its own specific type of iterator. Few Examples:

|  |  |
| --- | --- |
| **vector<int>::iterator** | **list<double>::iterator** |
| **set<std::string>::iterator** | **map<int, std::string>::iterator** |

Available functions in the std::string class which can be used for string manipulations.

**String Iteration**

****

OP



**String Length and Capacity**

Size: size refers to the number of characters in the string.

Capacity:

* The capacity of a container refers to the amount of memory that has been allocated for it, which determines how many elements it can hold efficiently.
* Capacity refers to the number of characters that the **string can hold without needing to reallocate memory**.
* When appending characters to a string, if the size exceeds the capacity, the string might need to be reallocated to accommodate the new characters.



OP

A black background with a white and blue light

Description automatically generated with medium confidence

**String Concatenation and Append**



OP

A black screen with white text

Description automatically generated

**insert**: It used to insert characters or a substring into a string at a specified position.



OP

A black background with white text

Description automatically generated

**find**:

A computer screen with text and images

Description automatically generated with medium confidence

OP



**substr**:

**A screen shot of a computer code

Description automatically generated**

OP

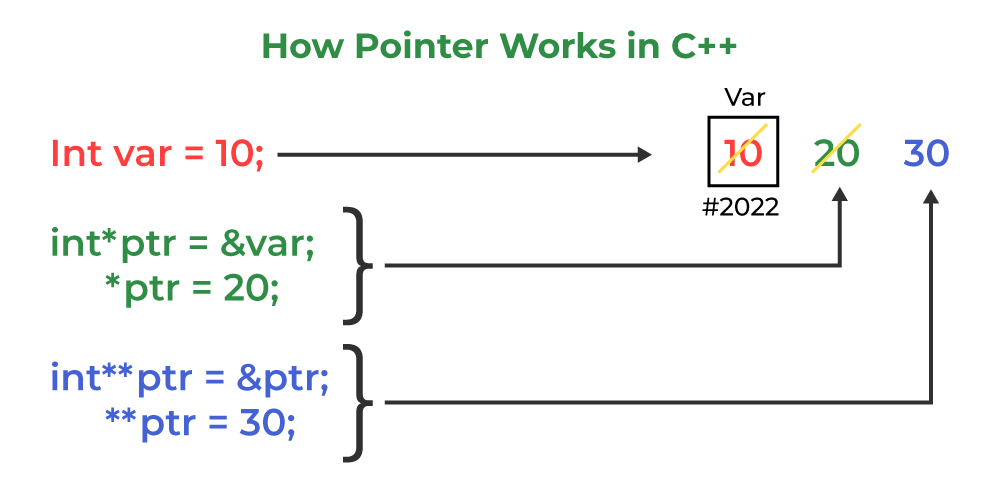


# Pointers & References in C++

## Pointers vs References

|  |  |
| --- | --- |
| **Pointers** | **Reference** |
| Pointer is a variable that **holds the memory address** of another variable. | A reference is an alias for an existing variable. It's essentially **another name** for the same memory location. |
| Pointers **can be reassigned** to point to different memory locations. | References must be initialized when declared, and they **cannot be reassigned** to refer to a different variable. |
| They can be **declared without initialization** and may contain garbage values. | They cannot be declared without initialization. |
| Pointer arithmetic is possible, allowing you to navigate through memory based on the size of the data type. | There is no pointer arithmetic with references. |
| Pointers can be made to **point to nullptr** to indicate they are not currently pointing to any valid memory location. | References **cannot be null or uninitialized**. They must always refer to a valid variable. |
| Pointers can be used for implementing data structures like linked lists, trees, and more. | References are useful when you want to avoid copying data. |

Pointers: Pointers is a variable that holds the memory address of another variable.



Use case:

* **Dynamic Memory Allocation**: This is useful when you need memory whose size is determined at runtime.

**A black background with white text

Description automatically generated**

* **Passing Parameters by Reference**: It allow the function to modify the original variable.

**A screen shot of a computer

Description automatically generated**

* **Pointers to Functions**: Pointers can be used to pass data structures or large objects efficiently to functions without making copies.

**A computer screen shot of a black screen

Description automatically generated**

* **Array Manipulation**

A computer code with white text

Description automatically generated

OP



* **Working with Hardware**

****

OP



* **Working with Dynamic Data Structures**: Pointers are essential when creating dynamic data structures like linked lists, trees, and graphs.

**A black background with white text

Description automatically generated**

* **Passing and Returning Multiple Values**

**A computer screen with white text and colorful text

Description automatically generated**

OP



* **Pointer Arithmetic**: Pointers allow you to perform arithmetic operations on memory addresses, which can be useful for traversing arrays or linked data structures.

**A black screen with white text

Description automatically generated**

Reference: References serve a different purpose than pointers. They **provide a convenient and safer way to work with values by creating an alias** for an existing variable.



Use case:

* **Function Parameter Passing**: References allow you to pass variables to functions by reference, enabling the function to modify the original variable's value.



* **Avoiding Copying**: Passing large objects or structures by reference avoids copying their contents.

A black background with white text

Description automatically generated

* **Range-Based Loops**: References are often used in range-based loops to directly access and modify elements in containers.

A black background with white text

Description automatically generated

* **Returning Multiple Values**: Functions can return multiple values through references, providing a convenient way to communicate results.

A black background with white text

Description automatically generated

## Types of Function Call (References v/s Pointers)

### Call-by-Value

A computer screen shot of a code

Description automatically generated

### Call-by-Reference with Pointer Arguments

A computer screen shot of a program code

Description automatically generated

### Call-by-Reference with Reference Arguments

A computer screen shot of a program code

Description automatically generated

## Array Name as Pointers