

C++ Pointers

Last Updated : 18 Mar, 2025



A **pointer** is a variable that stores the address of another variable. Pointers can be used with any data type, including basic types (e.g., int, char), arrays, and even user-defined types like classes and structures.

Create Pointer

A pointer can be declared in the same way as any other variable but with an **asterisk symbol (*)** as shown:

```
data_type* name
```



Here, **data_type** is the type of data that the pointer is pointing to, and **name** is the name assigned to the pointer. The ***** **symbol** is also called **dereference operator**.

Example:

```
int* ptr;
```



In the above statement, we create a pointer **ptr** that can store the address of an integer data. It is pronounced as “**Pointer to Integer**” or “**Integer Pointer**”

Assign Address

The **addressof operator (&)** determines the address of any variable in C++. This address can be assigned to the pointer variable to initialize it.

Example:

```
int val = 22;  
int* ptr = &val;
```



In the above statement, pointer **ptr** store the address of variable **val** using address-of operator (**&**). The pointer and the variable should be of same type, otherwise type mismatch error occurs.

Dereferencing

The process of accessing the value present at the memory address pointed by the pointer is called **dereferencing**. This is done with the help of dereferencing operator as shown:

Example:

```
▸ {...}
    // Access value using (*)
    // operator
    cout << *ptr;
▸ {...}
```

Output

10

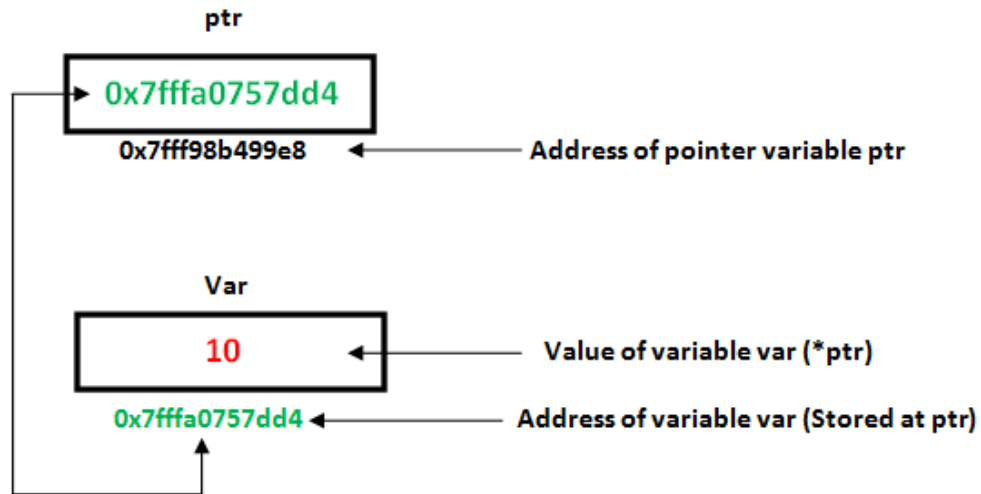
Directly accessing the pointer will just give us the address that is stored in the pointer.

```
▸ {...}
    // Access the address value
    cout << ptr;
▸ {...}
```

Output

0x7fffa0757dd4

The below image demonstrates how pointer works.



Modify Address

The pointer can be modified to point to another memory address if its is of the same type.

Example:

```

▶ {...}

int a = 10;
int b = 99;

int *ptr = &a;
cout << *ptr << endl;

// Changing the address stored
ptr = &b;
cout << *ptr;

▶ {...}

```

Output

```

10
99

```

Size of Pointers

The size of pointer in a system is equal for every pointer no matter what type of data it is pointing to. It does not depend on the type, but on operating system and CPU architecture. The size of pointers in C++ is

- **8 bytes** for a **64-bit System**
- **4 bytes** for a **32-bit System**

The logic is simple: pointers store the addresses of the memory and in a computer, the maximum width of a memory address is determined by the CPU architecture. For example, for a 64-bit CPU, the address will always be 64-bit wide. This can be verified using [sizeof operator](#).

```
▸ {...}
    int *ptr1;
    char *ptr2;

    // Finding size using sizeof()
    cout << sizeof(ptr1) << endl;
    cout << sizeof(ptr2);

▸ {...}
```

Output

```
8
8
```

As we can see, both the pointers have same size. It's a 64-bit system, so the size is 8 bytes.

Special Types of Pointers

There are 4 special types of pointers that used or referred to in different contexts:

Wild Pointer

When a pointer is created, it just contains a random address that may or may not be valid. This type of pointer is called **wild pointer**.

```
▸ {...}
    // Wild pointer
    int *ptr;

▸ {...}
```

Dereferencing this pointer may lead to errors such as segmentation faults. So, it is always recommended to initialize a pointer.

NULL Pointer

A [NULL pointer](#) is a pointer that does not point to any valid memory location but NULL. It is often used to initialize a pointer when you do not want it to point to any object.

```
▸ {...}
    // NULL pointer
    int *ptr = NULL;
▸ {...}
```



Void Pointers

This is a special type of pointer available in C++ which represents the absence of type. [Void pointers](#) are pointers that point to a value that has no particular type. It also means that they can point to any data type.

But as the datatype is not known, the compiler also does not know how many bytes to access when the void pointer is dereferenced, so these pointers cannot be directly dereferenced. They have to be first converted into some other pointer type that points to a concrete data type before being dereferenced.

Dangling Pointer

A [dangling pointer](#) is a pointer that was pointing to the valid memory location earlier, but that memory has been taken from the program, and it is still pointing to it. This pointer behaves like a wild pointer and may lead to errors.

Pointer and Arrays

In C++, [pointers and arrays](#) are closely related. An array name acts like a constant pointer to the first element of the array. If we assign this value to a non-constant pointer of the same type, then we can access the elements of the array with this pointer using pointer arithmetic.

Example:

```
▸ {...}
    int arr[5] = {10, 20, 30, 40, 50};

    // Creating a pointer to the first element
    int *ptr1 = &arr[0];
```



```
// Creating another pointer to array
int *ptr2 = arr;

// Accessing
cout << *ptr1 << endl;
cout << *ptr2;

▸ {...}
```

Output

```
10
10
```

Pointer Arithmetic

[Pointer arithmetic](#) refers to the operations that C++ allows on the pointers.

They include:

- Incrementing and Decrementing
- Addition of Integer
- Subtraction of Integer
- Subtraction of Two Pointers of the Same Type
- Comparison of Pointers and NULL

Pointer To Pointer (Double Pointer)

We can also have pointers that point to other pointers. This is called a [double pointer](#), and it is useful when working with multi-level data structures like arrays of pointers or dynamic memory management.

Example:

```
▸ {...}

int var = 10;

// Store the address of
// var variable
int* ptr1 = &var;
int** ptr2 = &ptr1;

// Access values using (*)
// operator
cout << *ptr1 << endl;
cout << **ptr2;

▸ {...}
```



Output

10
10

Pointer to Functions

In C++, a [function pointer](#) is used to point to functions, similar to how pointers are used to point to variables. It allows you to save the address of a function. Function pointers can be used to call a function indirectly, or they can be passed as arguments to another function, enabling dynamic function invocation and flexibility in function handling.

Smart Pointers

A [smart pointer](#) is a wrapper class around a pointer that overloads operators like * and ->. Smart pointer objects resemble normal pointers, they have the added functionality of automatically deallocating and freeing the memory of the object when it is no longer needed, unlike regular pointers.

Pointer vs Reference

Understanding the [differences between pointers and references in C++](#). Both are used to refer to other variables, but they operate in different ways and have different behavior.

Aspect	Pointer	Reference
Initialization	A pointer can be initialized after declaration.	A reference must be initialized at the time of declaration.
Nullability	A pointer can be assigned NULL or nullptr.	A reference cannot be null, it must always refer to a valid object.

Aspect	Pointer	Reference
Reassignment	A pointer can be reassigned to point to different objects.	A reference cannot be reassigned once it is bound to an object.

Uses of Pointers

Pointers are a useful concept in C++ that allow direct access to memory addresses, providing greater control over memory and data manipulation.

Below are some primary [uses of pointers in C++](#):

- **[Dynamic Memory Allocation](#)**: Pointers allow memory to be allocated dynamically at runtime using operators like new and delete. This enables the creation of objects or arrays whose sizes are determined during execution.
- **Implementing Data Structures**: Pointers are used to implementing complex data structures such as linked lists, trees, and graphs, where elements are dynamically allocated and linked together.
- **[Pass Arguments by Pointer](#)**: Pass the argument with their address and make changes in their value using pointer. So that the values get changed into the original argument.

[Comment](#)[More info](#) ▼[Advertise with us](#)[Next Article](#) >

Dangling, Void , Null and Wild
Pointers in C

Similar Reads

C++ Programming Language

C++ is a computer programming language developed by Bjarne Stroustrup as an extension of the C language. It is known for its fast speed, low level memory management and is often taught as first programming...

🕒 5 min read

C++ Overview



C++ Basics



C++ Variables and Constants



C++ Data Types and Literals



C++ Operators



C++ Input/Output



C++ Control Statements



C++ Functions



C++ Pointers and References



Pointers and References in C++

In C++ pointers and references both are mechanisms used to deal with memory, memory address, and data in a program. Pointers are used to store the memory address of another variable whereas references are us...

🕒 5 min read

C++ Pointers

A pointer is a variable that stores the address of another variable. Pointers can be used with any data type, including basic types (e.g., int, char), arrays, and even user-defined types like classes and structures. Create...

🕒 9 min read

Dangling, Void , Null and Wild Pointers in C

In C programming pointers are used to manipulate memory addresses, to store the address of some variable or memory location. But certain situations and characteristics related to pointers become challenging in...

🕒 6 min read

Applications of Pointers in C

Pointers in C are variables that are used to store the memory address of another variable. Pointers allow us to efficiently manage the memory and hence optimize our program. In this article, we will discuss some of...

 4 min read

Understanding nullptr in C++

Consider the following C++ program that shows problem with NULL (need of nullptr) [GFGTABS] CPP // C++ program to demonstrate problem with NULL #include <bits/stdc++.h> using namespace std; // function with...

 3 min read

References in C++

In C++, a reference works as an alias for an existing variable, providing an alternative name for it and allowing you to work with the original data directly. Example: [GFGTABS] C++ #include <iostream> using...

 6 min read

Can References Refer to Invalid Location in C++?

Reference Variables: You can create a second name for a variable in C++, which you can use to read or edit the original data contained in that variable. While this may not sound appealing at first, declaring a referenc...

 2 min read

Pointers vs References in C++

Prerequisite: Pointers, References C and C++ support pointers, which is different from most other programming languages such as Java, Python, Ruby, Perl and PHP as they only support references. But...

 5 min read

Passing By Pointer vs Passing By Reference in C++

In C++, we can pass parameters to a function either by pointers or by reference. In both cases, we get the same result. So, what is the difference between Passing by Pointer and Passing by Reference in C++? Let's...

 5 min read

When do we pass arguments by pointer?

In C, the pass-by pointer method allows users to pass the address of an argument to the function instead of the actual value. This allows programmers to change the actual data from the function and also improve th...

 5 min read[C++ Data Types](#)[C++ Input/Output](#)[C++ Arrays](#)[C++ Pointers](#)[C++ OOPs](#)[C++ STL](#)[C++ In](#)[Sign In](#)

(201305)

**Registered Address:**

K 061, Tower K, Gulshan Vivante
Apartment, Sector 137, Noida, Gautam
Buddh Nagar, Uttar Pradesh, 201305

[Advertise with us](#)**Company**

[About Us](#)
[Legal](#)
[Privacy Policy](#)
[In Media](#)
[Contact Us](#)
[Advertise with us](#)
[GFG Corporate Solution](#)
[Placement Training Program](#)

DSA

[Data Structures](#)
[Algorithms](#)
[DSA for Beginners](#)
[Basic DSA Problems](#)
[DSA Roadmap](#)
[Top 100 DSA Interview Problems](#)
[DSA Roadmap by Sandeep Jain](#)
[All Cheat Sheets](#)

Web Technologies

[HTML](#)
[CSS](#)
[JavaScript](#)
[TypeScript](#)
[ReactJS](#)
[NextJS](#)
[Bootstrap](#)
[Web Design](#)

Computer Science**Languages**

[Python](#)
[Java](#)
[C++](#)
[PHP](#)
[GoLang](#)
[SQL](#)
[R Language](#)
[Android Tutorial](#)
[Tutorials Archive](#)

Data Science & ML

[Data Science With Python](#)
[Data Science For Beginner](#)
[Machine Learning](#)
[ML Maths](#)
[Data Visualisation](#)
[Pandas](#)
[NumPy](#)
[NLP](#)
[Deep Learning](#)

Python Tutorial

[Python Programming Examples](#)
[Python Projects](#)
[Python Tkinter](#)
[Python Web Scraping](#)
[OpenCV Tutorial](#)
[Python Interview Question](#)
[Django](#)

DevOps

Operating Systems
Computer Network
Database Management System
Software Engineering
Digital Logic Design
Engineering Maths
Software Development
Software Testing

System Design

High Level Design
Low Level Design
UML Diagrams
Interview Guide
Design Patterns
OOAD
System Design Bootcamp
Interview Questions

School Subjects

Mathematics
Physics
Chemistry
Biology
Social Science
English Grammar
Commerce
World GK

Git
Linux
AWS
Docker
Kubernetes
Azure
GCP
DevOps Roadmap

Interview Preparation

Competitive Programming
Top DS or Algo for CP
Company-Wise Recruitment Process
Company-Wise Preparation
Aptitude Preparation
Puzzles

GeeksforGeeks Videos

DSA
Python
Java
C++
Web Development
Data Science
CS Subjects

@GeeksforGeeks, Sanchhaya Education Private Limited, All rights reserved

Do Not Sell or Share My Personal Information