C Pointers

Last Updated : 13 May, 2025

A **pointer** is a variable that stores the **memory address** of another variable. Instead of holding a direct value, it has the address where the value is stored in memory. This allows us to manipulate the data stored at a specific memory location without actually using its variable. It is the backbone of low-level memory manipulation in C.

Declare a Pointer

A pointer is declared by specifying its name and type, just like simple variable declaration but with an **asterisk (*)** symbol added before the pointer's name.

```
data_type* name
```

Here, **data_type** defines the type of data that the pointer is pointing to. An integer type pointer can only point to an integer. Similarly, a pointer of float type can point to a floating-point data, and so on.

Example:

```
int *ptr;
```

In the above statement, pointer **ptr** can store the address of an integer. It is pronounced as pointer to integer.

Initialize the Pointer

Pointer initialization means assigning some address to the pointer variable. In C, the **(&) addressof operator** is used to get the memory address of any variable. This memory address is then stored in a pointer

```
int var = 10;

// Initializing ptr
int *ptr = &var;
```

In the above statement, pointer **ptr** store the address of variable **x** which was determined using address-of operator **(&).**

> ***Note:*** *We can also declare and initialize the pointer in a single step. This is called* ***pointer definition.***

Dereference a Pointer

Accessing the pointer directly will just give us the address that is stored in the pointer. For example,

```c
#include <stdio.h>

int main() {
    int var = 10;

    // Store address of var variable
    int* ptr = &var;

    // Directly accessing ptr
    printf("%d", ptr);

    return 0;
}
```

Output

```
0x7fffa0757dd4
```

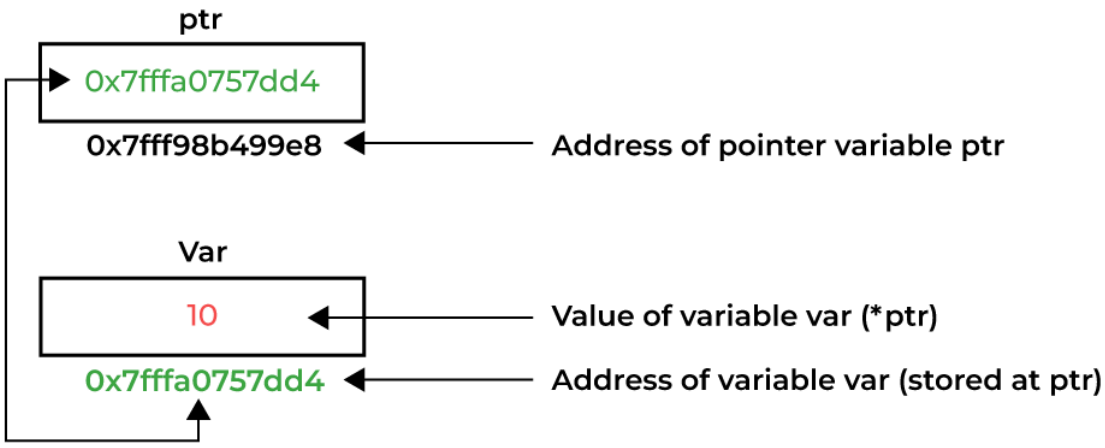
This hexadecimal integer (starting with 0x) is the memory address.

We have to first **dereference** the pointer to access the value present at the memory address. This is done with the help of **dereferencing operator(*)** (same operator used in declaration).

```
    int var = 10;

    // Store address of var variable
    int* ptr = &var;

    // Dereferencing ptr to access the value
    printf("%d", *ptr);

    return 0;
}
```

Output

```
10
```



__Note__: Earlier, we used %d for printing pointers, but C provides a separate format specifier __%p__ for printing pointers.

Size of Pointers

The **size of a pointer in C** depends on the **architecture (bit system)** of the machine, **not the data type** it points to.

- On a **32-bit system**, all pointers typically occupy **4 bytes**.
- On a **64-bit system**, all pointers typically occupy **8 bytes**.

The size remains **constant regardless of the data type** (`int*`, `char*`,

```c
int main() {
    int *ptr1;
    char *ptr2;

    // Finding size using sizeof()
    printf("%zu\n", sizeof(ptr1));
    printf("%zu", sizeof(ptr2));

    return 0;
}
```

Output

```
8
8
```

The reason for the same size is that the pointers store the memory addresses, no matter what type they are. As the space required to store the addresses of the different memory locations is the same, the memory required by one pointer type will be equal to the memory required by other pointer types.

> ***Note:*** *The actual size of the pointer may vary depending on the* ***compiler and system architecture****, but it is always* ***uniform across all data types*** *on the same system.*

Special Types of Pointers

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

NULL Pointer

The NULL Pointers are those pointers that do not point to any memory location. They can be created by assigning **NULL** value to the pointer. A pointer of any type can be assigned the NULL value.

```
5       int *ptr = NULL;
6
7       return 0;
8   }
```

NULL pointers are generally used to represent the absence of any address. This allows us to check whether the pointer is pointing to any valid memory location by checking if it is equal to NULL.

Void Pointer

The void pointers in C are the pointers of type **void**. It means that they do not have any associated data type. They are also called **generic pointers** as they can point to any type and can be typecasted to any type.

```
1   #include <stdio.h>
2
3   int main() {
4       // Void pointer
5       void *ptr;
6
7       return 0;
8   }
```

Wild Pointers

The wild pointers are pointers that have not been initialized with something yet. These types of C-pointers can cause problems in our programs and can eventually cause them to crash. If values are updated using wild pointers, they could cause data abort or data corruption.

```
1   #include <stdio.h>
2
3   int main() {
4
```

```
9   }
```

Dangling Pointer

A pointer pointing to a memory location that has been deleted (or freed) is called a dangling pointer. Such a situation can lead to unexpected behavior in the program and also serve as a source of bugs in C programs.

```c
#include <stdio.h>
#include <stdlib.h>

int main() {
    int* ptr = (int*)malloc(sizeof(int));

    // After below free call, ptr becomes a dangling pointer
    free(ptr);
    printf("Memory freed\n");

    // removing Dangling Pointer
    ptr = NULL;

    return 0;
}
```

Output

```
Memory freed
```

C Pointer Arithmetic

The pointer arithmetic refers to the arithmetic operations that can be performed on a pointer. It is slightly different from the ones that we generally use for mathematical calculations as only a limited set of operations can be performed on pointers. These operations include:

- Comparing/Assigning Two Pointers of Same Type
- Comparing/Assigning with NULL

C Pointers and Arrays

In C programming language, pointers and arrays are closely related. An array name acts like a pointer constant. The value of this pointer constant is the address of the first element. For example, if we have an array named **val,** then **val** and **&val[0]** can be used interchangeably.

If we assign this value to a non-constant pointer to array of the same type, then we can access the elements of the array using this pointer. Not only that, as the array elements are stored continuously, we can use pointer arithmetic operations such as increment, decrement, addition, and subtraction of integers on pointer to move between array elements.

This concept is not limited to the one-dimensional array, we can refer to a multidimensional array element as well using pointers.

Constant Pointers

In **constant pointers**, the memory address stored inside the pointer is constant and cannot be modified once it is defined. It will always point to the same memory address.

Example:

```c
#include <stdio.h>

int main() {
    int a = 90;
    int b = 50;

    // Creating a constant pointer
    int* const ptr = &a;

    // Trying to reassign it to b
    ptr = &b;

    return 0;
```

Output

```
solution.c: In function ‘main’:
solution.c:11:9: error: assignment of read-only variable
‘ptr’
   11 |     ptr = &b;
      |         ^
```

We can also create a pointer to constant or even constant pointer to constant. Refer to this article to know more - Constant pointer, Pointers to Constant and Constant Pointers to Constant

Pointer to Function

A **function pointer** is a type of pointer that stores the address of a function, allowing functions to be passed as arguments and invoked dynamically. It is useful in techniques such as callback functions, event-driven programs.

Example:

```c
#include <stdio.h>

int add(int a, int b) {
    return a + b;
}

int main() {

    // Declare a function pointer that matches
    // the signature of add() fuction
    int (*fptr)(int, int);

    // Assign address of add()
    fptr = &add;

    // Call the function via ptr
    printf("%d", fptr(10, 5));
```

```
20  }
```

Output

```
15
```

Multilevel Pointers

In C, we can create **multi-level pointers** with any number of levels such as – ***ptr3, ****ptr4, ******ptr5 and so on. Most popular of them is **double pointer** (pointer to pointer). It stores the memory address of another pointer. Instead of pointing to a data value, they point to another pointer.

Example:

```
#include <stdio.h>

int main() {
    int var = 10;

    // Pointer to int
    int *ptr1 = &var;

    // Pointer to pointer (double pointer)
    int **ptr2 = &ptr1;

    // Accessing values using all three
    printf("var: %d\n", var);
    printf("*ptr1: %d\n", *ptr1);
    printf("**ptr2: %d", **ptr2);

    return 0;
}
```

Output

```
var: 10
```

Uses of Pointers in C

The C pointer is a very powerful tool that is widely used in C programming to perform various useful operations. It is used to achieve the following functionalities in C:

- Pass Arguments by Pointers
- Accessing Array Elements
- Return Multiple Values from Function
- Dynamic Memory Allocation
- Implementing Data Structures
- In System-Level Programming where memory addresses are useful.
- To use in Control Tables.

Advantages of Pointers

Following are the major advantages of pointers in C:

- Pointers are used for dynamic memory allocation and deallocation.
- An Array or a structure can be accessed efficiently with pointers
- Pointers are useful for accessing memory locations.
- Pointers are used to form complex data structures such as linked lists, graphs, trees, etc.
- Pointers reduce the length of the program and its execution time as well.

Issues with Pointers

Pointers are vulnerable to errors and have following disadvantages:

- Memory corruption can occur if an incorrect value is provided to pointers.
- Pointers are a little bit complex to understand.
- Pointers are majorly responsible for memory leaks in C.
- Accessing using pointers are comparatively slower than variables in C.
- Uninitialized pointers might cause a segmentation fault.

Comment More info

Campus Training Program

Next Article

Pointer Arithmetics in C with Examples

Similar Reads

C Identifiers

In C programming, identifiers are the names used to identify variables, functions, arrays, structures, or any other user-defined items. It is a name...

15+ min read

Tokens in C

In C programming, tokens are the smallest units in a program that have meaningful representations. Tokens are the building blocks of a C...

15+ min read

strcspn() in C

The C library function strcspn() calculates the length of the number of characters before the 1st occurrence of character present in both the...

15 min read

Strings in C

A String in C programming is a sequence of characters terminated with a null character '\0'. The C String is work as an array of characters. The...

15+ min read

puts() in C

In C programming language, puts() is a function defined in header <stdio.h> that prints strings character by character until the NULL...

strpbrk() in C

This function finds the first character in the string s1 that matches any character specified in s2 (It excludes terminating null-characters). Syntax...

12 min read

snprintf() in C

In C, snprintf() function is a standard library function that is used to print the specified string till a specified length in the specified format. It is...

15+ min read

tolower() Function in C

tolower() function in C is used to convert the uppercase alphabet to the lowercase alphabet. It does not affect characters other than uppercase...

7 min read

isless() in C/C++

In C++, isless() is a predefined function used for mathematical calculations. math.h is the header file required for various mathematical...

11 min read

Format Specifiers in C

The format specifier in C is used to tell the compiler about the type of data to be printed or scanned in input and output operations. They always sta...

15+ min read

Web Scraping
OpenCV Tutorial
Python Interview Question

DevOps

Git
AWS
Docker
Kubernetes
Azure
GCP
DevOps Roadmap

School Subjects

Mathematics
Physics
Chemistry
Biology
Social Science
English Grammar

Preparation Corner

Company-Wise Recruitment Process
Aptitude Preparation
Puzzles
Company-Wise Preparation

Machine Learning/Data Science

Complete Machine Learning & Data Science Program - [LIVE]
Data Analytics Training using Excel, SQL, Python & PowerBI - [LIVE]
Data Science Training Program - [LIVE]
Data Science Course with IBM Certification

Clouds/Devops

DevOps Engineering
AWS Solutions Architect Certification
Salesforce Certified Administrator Course

Software Engineering
Digital Logic Design
Engineering Maths

System Design

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

Databases

SQL
MYSQL
PostgreSQL
PL/SQL
MongoDB

More Tutorials

Software Development
Software Testing
Product Management
Project Management
Linux
Excel
All Cheat Sheets

Programming Languages

C Programming with Data Structures
C++ Programming Course
Java Programming Course
Python Full Course

GATE 2026

GATE CS Rank Booster
GATE DA Rank Booster
GATE CS & IT Course - 2026
GATE DA Course 2026
GATE Rank Predictor