C++ Pointers

Difficulty Level : Easy   ●   Last Updated : 25 Oct, 2022

Pointers are symbolic representations of addresses. They enable programs to simulate call-by-reference as well as to create and manipulate dynamic data structures. Iterating over elements in arrays or other data structures is one of the main use of pointers.
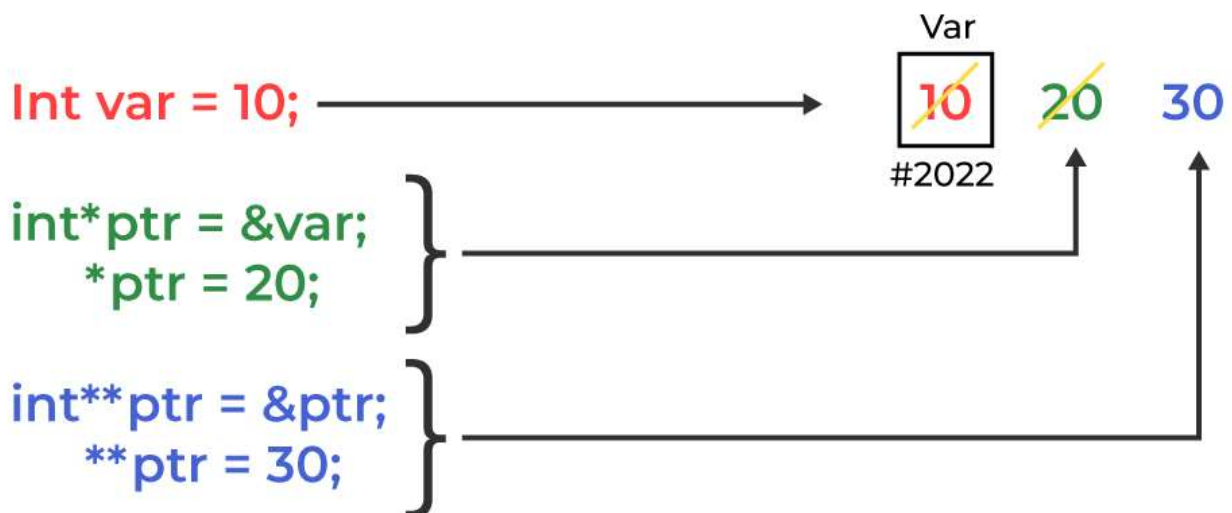
The address of the variable you're working with is assigned to the pointer variable that points to the same data type (such as an int or string).

Syntax:

```
datatype *var_name;
int *ptr;    // ptr can point to an address which holds int data
```

How Pointer Works in C++

How to use a pointer?

returns the address of that variable.
- Accessing the value stored in the address using unary operator (*) which returns the value of the variable located at the address specified by its operand.

The reason we associate data type with a pointer is **that it knows how many bytes the data is stored in**. When we increment a pointer, we increase the pointer by the size of the data type to which it points.

C++
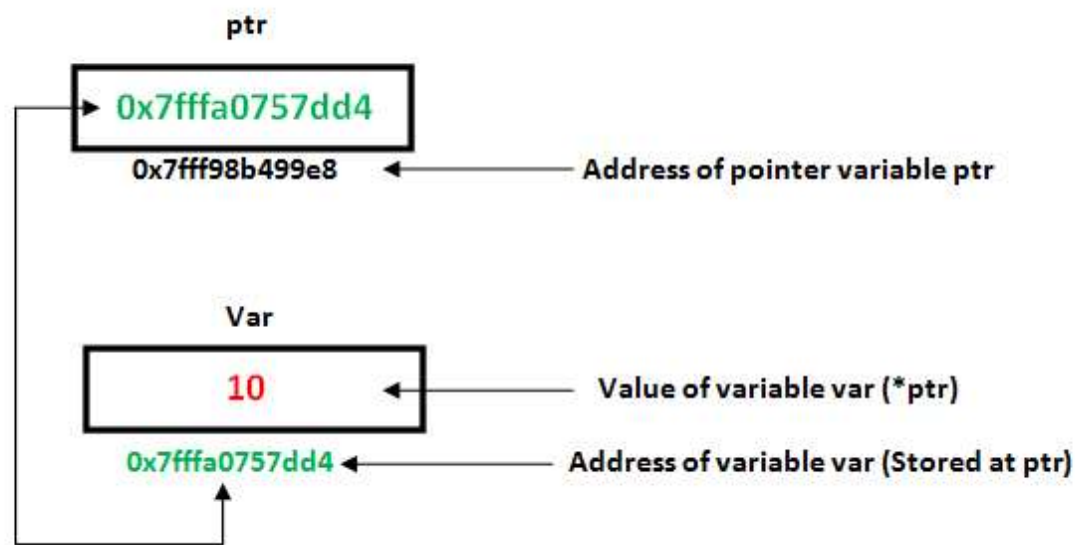
```cpp
// C++ program to illustrate Pointers

#include <bits/stdc++.h>
using namespace std;
void geeks()
{
    int var = 20;

    // declare pointer variable
```

Start Your Coding Journey Now!

```cpp
    // note that data type of ptr and var must be same
    ptr = &var;
```

```cpp
    cout << "Value at ptr = " << ptr << "\n";
    cout << "Value at var = " << var << "\n";
    cout << "Value at *ptr = " << *ptr << "\n";
}
// Driver program
int main()
{
    geeks();
    return 0;
}
```

Output

```
Value at ptr = 0x7ffe454c08cc
Value at var = 20
Value at *ptr = 20
```

References and Pointers

There are 3 ways to pass C++ arguments to a function:

- Call-By-Value
- Call-By-Reference with a Pointer Argument
- Call-By-Reference with a Reference Argument

Start Your Coding Journey Now!

C++

Read   Discuss   Courses   Practice   Video

```cpp
#include <bits/stdc++.h>
using namespace std;

// Pass-by-Value
int square1(int n)
{
    // Address of n in square1() is not the same as n1 in
    // main()
    cout << "address of n1 in square1(): " << &n << "\n";

    // clone modified inside the function
    n *= n;
    return n;
}
// Pass-by-Reference with Pointer Arguments
void square2(int* n)
{
    // Address of n in square2() is the same as n2 in main()
    cout << "address of n2 in square2(): " << n << "\n";

    // Explicit de-referencing to get the value pointed-to
    *n *= *n;
}
// Pass-by-Reference with Reference Arguments
void square3(int& n)
{
    // Address of n in square3() is the same as n3 in main()
    cout << "address of n3 in square3(): " << &n << "\n";

    // Implicit de-referencing (without '*')
    n *= n;
}
void geeks()
{
    // Call-by-Value
    int n1 = 8;
    cout << "address of n1 in main(): " << &n1 << "\n";
    cout << "Square of n1: " << square1(n1) << "\n";
    cout << "No change in n1: " << n1 << "\n";

    // Call-by-Reference with Pointer Arguments
    int n2 = 8;
```

Start Your Coding Journey Now!

```cpp
    cout << "Square of n2: " << n2 << "\n";
    cout << "Change reflected in n2: " << n2 << "\n";
```

Read   Discuss   Courses   Practice   Video

```cpp
    int n3 = 8;
    cout << "address of n3 in main(): " << &n3 << "\n";
    square3(n3);
    cout << "Square of n3: " << n3 << "\n";
    cout << "Change reflected in n3: " << n3 << "\n";
}
// Driver program
int main() { geeks(); }
```

Output

```
 address of n1 in main(): 0x7fffa7e2de64
 address of n1 in square1(): 0x7fffa7e2de4c
 Square of n1: 64
 No change in n1: 8
 address of n2 in main(): 0x7fffa7e2de68
 address of n2 in square2(): 0x7fffa7e2de68
 Square of n2: 64
 Change reflected in n2: 64
 address of n3 in main(): 0x7fffa7e2de6c
 address of n3 in square3(): 0x7fffa7e2de6c
 Square of n3: 64
 Change reflected in n3: 64
```

In C++, by default arguments are passed by value and the changes made in the called function will not reflect in the passed variable. The changes are made into a clone made by the called function. If wish to modify the original copy directly (especially in passing huge object or array) and/or avoid the overhead of cloning, we use pass-by-reference. Pass-by-Reference with Reference Arguments does not require any clumsy syntax for referencing and dereferencing.

- [Function pointers in C](#)
- [Pointer to a Function](#)

Start Your Coding Journey Now!

An array name contains the address of the first element of the array which acts like a

interchangeably.

C++

```cpp
// C++ program to illustrate Array Name as Pointers
#include <bits/stdc++.h>
using namespace std;
void geeks()
{
    // Declare an array
    int val[3] = { 5, 10, 20 };

    // declare pointer variable
    int* ptr;

    // Assign the address of val[0] to ptr
    // We can use ptr=&val[0];(both are same)
    ptr = val;
    cout << "Elements of the array are: ";
    cout << ptr[0] << " " << ptr[1] << " " << ptr[2];
}
// Driver program
int main() { geeks(); }
```

Output

```
Elements of the array are: 5 10 20
```

val[0]	val[1]	val[2]
5	10	15
ptr[0]	ptr[1]	ptr[2]

If pointer ptr is sent to a function as an argument, the array val can be accessed in a similar fashion. Pointer vs Array

Start Your Coding Journey Now!

A limited set of arithmetic operations can be performed on pointers which are:

- an integer may be added to a pointer (+ or +=)
- an integer may be subtracted from a pointer (– or -=)
- difference between two pointers (p1-p2)

(**Note:** Pointer arithmetic is meaningless unless performed on an array.)

C++

```cpp
// C++ program to illustrate Pointer Arithmetic
#include <bits/stdc++.h>
using namespace std;
void geeks()
{
    // Declare an array
    int v[3] = { 10, 100, 200 };

    // declare pointer variable
    int* ptr;

    // Assign the address of v[0] to ptr
    ptr = v;

    for (int i = 0; i < 3; i++) {
        cout << "Value at ptr = " << ptr << "\n";
        cout << "Value at *ptr = " << *ptr << "\n";

        // Increment pointer ptr by 1
        ptr++;
    }
}

// Driver program
int main() { geeks(); }
```

Output

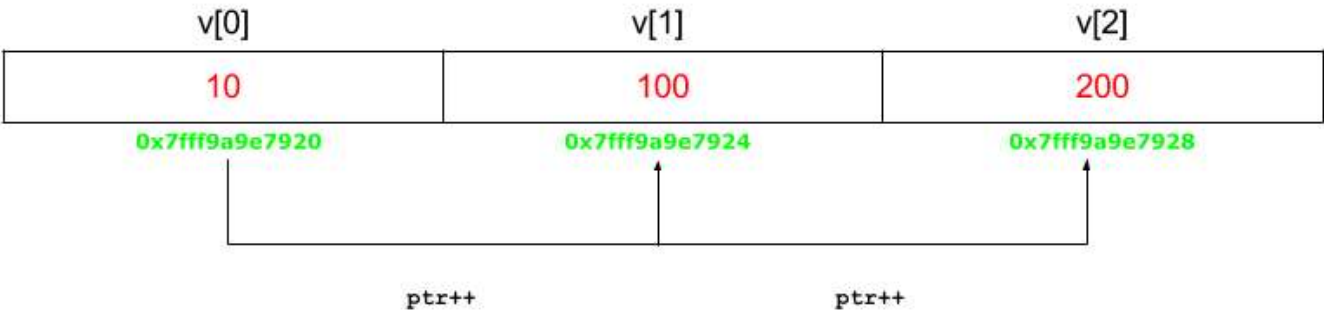```
Value at ptr = 0x7ffe5a2d8060
```

Value at *ptr = 100
Value at ptr = 0x7ffe5a2d8068

## Advanced Pointer Notation

Consider pointer notation for the two-dimensional numeric arrays. consider the following declaration

```
int nums[2][3]  =  { { 16, 18, 20 }, { 25, 26, 27 } };
```

In general, nums[i] [j] is equivalent to *(*(nums+i)+j)

Pointer Notation	Array Notation	Value
*(*nums)	nums[0][0]	16
*(*nums+1)	nums[0][1]	18
*(*nums+2)	nums[0][2]	20
((nums + 1))	nums[1][0]	25
((nums + 1)+1)	nums[1][1]	26
((nums + 1)+2)	nums[1][2]	27

Pointers and String literals

String literals are arrays containing null-terminated character sequences. String literals are arrays of type character plus terminating null-character, with each of the elements

Start Your Coding Journey Now!

ts

first element is assigned to ptr. If we imagine that "geek" is stored at the memory locations that start at address 1800, we can represent the previous declaration as:

'g'	'e'	'e'	'k'	'\0'
1800	1801	1802	1803	1804

As pointers and arrays behave in the same way in expressions, ptr can be used to access the characters of a string literal. For example:

```
char x = *(ptr+3);
char y = ptr[3];
```

Here, both x and y contain k stored at 1803 (1800+3).

Pointers to pointers

In C++, we can create a pointer to a pointer that in turn may point to data or another pointer. The syntax simply requires the unary operator (*) for each level of indirection while declaring the pointer.

```
char a;
char *b;
char ** c;
a = 'g';
b = &a;
c = &b;
```

Here b points to a char that stores 'g' and c points to the pointer b.

Void Pointers

This is a special type of pointer available in C++ which represents the absence of type

Start Your Coding Journey Now!

undetermined length and undetermined dereferencing properties). This means that void

transformed into some other pointer type that points to a concrete data type before being dereferenced.

C++

```cpp
// C++ program to illustrate Void Pointer
#include <bits/stdc++.h>
using namespace std;
void increase(void* data, int ptrsize)
{
    if (ptrsize == sizeof(char)) {
        char* ptrchar;

        // Typecast data to a char pointer
        ptrchar = (char*)data;

        // Increase the char stored at *ptrchar by 1
        (*ptrchar)++;
        cout << "*data points to a char"
             << "\n";
    }
    else if (ptrsize == sizeof(int)) {
        int* ptrint;

        // Typecast data to a int pointer
        ptrint = (int*)data;

        // Increase the int stored at *ptrchar by 1
        (*ptrint)++;
        cout << "*data points to an int"
             << "\n";
    }
}
void geek()
{
    // Declare a character
    char c = 'x';

    // Declare an integer
    int i = 10;
```

Start Your Coding Journey Now!

```
    increase(&c, sizeof(c));
    cout << "The new value of c is: " << c << "\n";
```

```
}
// Driver program
int main() { geek(); }
```

Output

```
*data points to a char
The new value of c is: y
*data points to an int
The new value of i is: 11
```

Invalid pointers

A pointer should point to a valid address but not necessarily to valid elements (like for arrays). These are called invalid pointers. Uninitialized pointers are also invalid pointers.

```
int *ptr1;
int arr[10];
int *ptr2 = arr+20;
```

Here, ptr1 is uninitialized so it becomes an invalid pointer and ptr2 is out of bounds of arr so it also becomes an invalid pointer. (Note: invalid pointers do not necessarily raise compile errors)

NULL Pointers

A null pointer is a pointer that point nowhere and not just an invalid address. Following are 2 methods to assign a pointer as NULL;

```
int *ptr1 = 0;
int *ptr2 = NULL;
```

Advantages of Pointers

Start Your Coding Journey Now!

- Pointers reduce the code and improve performance. They are used to retrieve strings, trees, arrays, structures, and functions.

| Read | Discuss | Courses | Practice | Video |

- In addition to this, pointers allow us to access a memory location in the computer's memory.

Related Articles:

- Opaque Pointer
- Near, Far and huge Pointers

Quizzes:

- Pointer Basics
- Advanced Pointer

This article is contributed by **Abhirav Kariya.** If you like GeeksforGeeks and would like to contribute, you can also write an article using write.geeksforgeeks.org. See your article appearing on the GeeksforGeeks main page and help other Geeks. Please write comments if you find anything incorrect, or if you want to share more information about the topic discussed above.

Like   160

Previous   Next

Related Articles

Start Your Coding Journey Now!

Read   Discuss   Courses   Practice   Video

2.   What are Wild Pointers? How can we avoid?

3.   Declare a C/C++ function returning pointer to array of integer pointers

4.   Smart Pointers in C++ and How to Use Them

5.   Pointers vs References in C++

6.   Computing Index Using Pointers Returned By STL Functions in C++

7.   Applications of Pointers in C/C++

8.   C++ Program to compare two string using pointers

9.   Program to Reverse a String using Pointers

10.   How to implement user defined Shared Pointers in C++

Article Contributed By :

GeeksforGeeks

Vote for difficulty

Current difficulty : Easy

| Easy | Normal | Medium | Hard | Expert |

Start Your Coding Journey Now!

Read   Discuss   Courses   Practice   Video

Article Tags :   cpp-pointer,   C++

Practice Tags :   CPP

Improve Article   Report Issue

GeeksforGeeks

A-143, 9th Floor, Sovereign Corporate Tower,
Sector-136, Noida, Uttar Pradesh - 201305

feedback@geeksforgeeks.org

Company

About Us

Careers

In Media

Contact Us

Privacy Policy

Copyright Policy

Advertise with us

News

Top News

Technology

Work & Career

Learn

DSA

Algorithms

Data Structures

SDE Cheat Sheet

Machine learning

CS Subjects

Video Tutorials

Courses

Languages

Python

Java

Start Your Coding Journey Now!

Finance   C#

Read   Discuss   Courses   Practice   Video

Web Development   ## Contribute

Web Tutorials   Write an Article

Django Tutorial   Improve an Article

HTML   Pick Topics to Write

JavaScript   Write Interview Experience

Bootstrap   Internships

ReactJS   Video Internship

NodeJS

Start Your Coding Journey Now!