



LEARN C PROGRAMMING

c programming language

C Programming Tutorial

C - Home

C - Overview

C - Environment Setup

C - Program Structure

C - Basic Syntax

C - Data Types

C - Variables

C - Constants

C - Storage Classes

C - Operators

C - Decision Making

C - Loops

C - Functions

C - Scope Rules

C - Arrays

C - Pointers

C - Strings

C - Structures

C - Unions

C - Bit Fields

C - Typedef

C - Input & Output

C - File I/O

C - Preprocessors

C - Header Files

C - Type Casting

C - Error Handling

C - Recursion

C - Variable Arguments

C - Memory Management

C - Command Line Arguments

C Programming Resources

C - Questions and Answers

C - Quick Guide

C - Useful Resources

C - Discussion

Selected Reading

## C - Pointer arithmetic

Advertisements

15 Day FREE Access  
to Analytics and Big Data videos and more.



START FREE TRIAL

Previous Page

Next Page

As explained in main chapter, C pointer is an address, which is a numeric value. Therefore, you can perform arithmetic operations on a pointer just as you can a numeric value. There are four arithmetic operators that can be used on pointers: ++, --, +, and -

To understand pointer arithmetic, let us consider that **ptr** is an integer pointer which points to the address 1000. Assuming 32-bit integers, let us perform the following arithmetic operation on the pointer:

```
ptr++
```

Now, after the above operation, the **ptr** will point to the location 1004 because each time ptr is incremented, it will point to the next integer location which is 4 bytes next to the current location. This operation will move the pointer to next memory location without impacting actual value at the memory location. If **ptr** points to a character whose address is 1000, then above operation will point to the location 1001 because next character will be available at 1001.

### Incrementing a Pointer

We prefer using a pointer in our program instead of an array because the variable pointer can be incremented, unlike the array name which cannot be incremented because it is a constant pointer. The following program increments the variable pointer to access each succeeding element of the array:

```
#include <stdio.h>

const int MAX = 3;

int main ()
{
    int var[] = {10, 100, 200};
    int i, *ptr;

    /* let us have array address in pointer */
    ptr = var;
    for ( i = 0; i < MAX; i++)
    {

        printf("Address of var[%d] = %x\n", i, ptr );
        printf("Value of var[%d] = %d\n", i, *ptr );

        /* move to the next location */
        ptr++;
    }
    return 0;
}
```

When the above code is compiled and executed, it produces result something as follows:

```
Address of var[0] = bf882b30
Value of var[0] = 10
Address of var[1] = bf882b34
Value of var[1] = 100
Address of var[2] = bf882b38
Value of var[2] = 200
```

### Decrementing a Pointer

The same considerations apply to decrementing a pointer, which decreases its value by the number of bytes of its data type as shown below:

```
#include <stdio.h>

const int MAX = 3;
```

WEAR YOUR SHINE  
by PC Jeweller

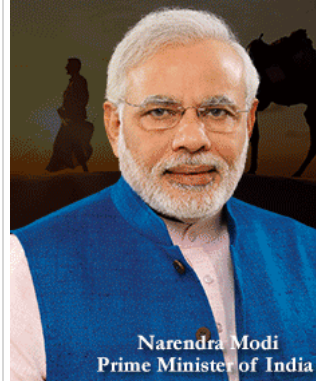
TRENDSETTING RING



BUY NOW

Share your experience  
& photos using  
**#IncredibleIndia**  
and I'll share it with  
the world.

View PM's Album



Narendra Modi  
Prime Minister of India



Become A  
Business Analyst

15 Day FREE Access  
to Analytics and Big Data  
videos and more.



START FREE TRIAL

Developer's Best Practices
Questions and Answers
Effective Resume Writing
Computer Glossary
Who is Who

```
int main ()
{
    int var[] = {10, 100, 200};
    int i, *ptr;

    /* let us have array address in pointer */
    ptr = &var[MAX-1];
    for ( i = MAX; i > 0; i--)
    {

        printf("Address of var[%d] = %x\n", i, ptr );
        printf("Value of var[%d] = %d\n", i, *ptr );

        /* move to the previous location */
        ptr--;
    }
    return 0;
}
```

When the above code is compiled and executed, it produces result something as follows:

```
Address of var[3] = bfedbcd8
Value of var[3] = 200
Address of var[2] = bfedbcd4
Value of var[2] = 100
Address of var[1] = bfedbcd0
Value of var[1] = 10
```

## Pointer Comparisons

Pointers may be compared by using relational operators, such as ==, <, and >. If p1 and p2 point to variables that are related to each other, such as elements of the same array, then p1 and p2 can be meaningfully compared.

The following program modifies the previous example one by incrementing the variable pointer so long as the address to which it points is either less than or equal to the address of the last element of the array, which is &var[MAX - 1]:

```
#include <stdio.h>

const int MAX = 3;

int main ()
{
    int var[] = {10, 100, 200};
    int i, *ptr;

    /* let us have address of the first element in pointer */
    ptr = var;
    i = 0;
    while ( ptr <= &var[MAX - 1] )
    {

        printf("Address of var[%d] = %x\n", i, ptr );
        printf("Value of var[%d] = %d\n", i, *ptr );

        /* point to the previous location */
        ptr++;
        i++;
    }
    return 0;
}
```

When the above code is compiled and executed, it produces result something as follows:

```
Address of var[0] = bfdbcb20
Value of var[0] = 10
Address of var[1] = bfdbcb24
Value of var[1] = 100
Address of var[2] = bfdbcb28
Value of var[2] = 200
```



Extras

## Mobile First



## About us

- ▶ Company
- ▶ Team
- ▶ Careers
- ▶ Privacy Policy
- ▶ Terms of use

## Extra Links

- ▶ Forums
- ▶ Articles
- ▶ Shared
- ▶ Seo Tools
- ▶ Contact

## Contact Us

📍 Address: 388-A , Road no 22, Jubilee Hills,  
Hyderabad Telangana, INDIA-500033

✉ Email: [Click Here](#)

🌐 Website: [www.tutorialspoint.com](http://www.tutorialspoint.com)



© Copyright 2015. All Rights Reserved.

[Write for us](#) [FAQ's](#) [Helping](#) [Contact](#)