Codeforwin

Learn C programming, Data Structures tutorials, exercises, examples, programs, hacks, tips and tricks online.

Pointer arithmetic in C programming

October 24, 2017 Pankaj C programming C, Pointer, Programming, Tutorial

Pointer is a variable that points to a memory location. Memory addresses are numeric value that ranges from zero to maximum memory size in bytes. These addresses can be manipulated like simple variables. You can increment, decrement, calculate or compare these addresses manually.

C language provides a set of operators to perform arithmetic and comparison of memory addresses. Pointer arithmetic and comparison in C is supported by following operators –

- Increment and decrement `++` and `--`
- Addition and Subtraction `+` and `-`
- Comparison `<` , `>` , `<=` , `>=` , `==` , `!=`

Pointer increment and decrement

Increment operator when used with a pointer variable returns next address pointed by the pointer. The next address returned is the sum of current pointed address and size of pointer data type.

> *Read more about, how to find size of a data type.*

Or in simple terms, incrementing a pointer will cause the pointer to point to a memory location skipping `N` bytes from current pointed memory location. Where `N` is size of pointer data type.

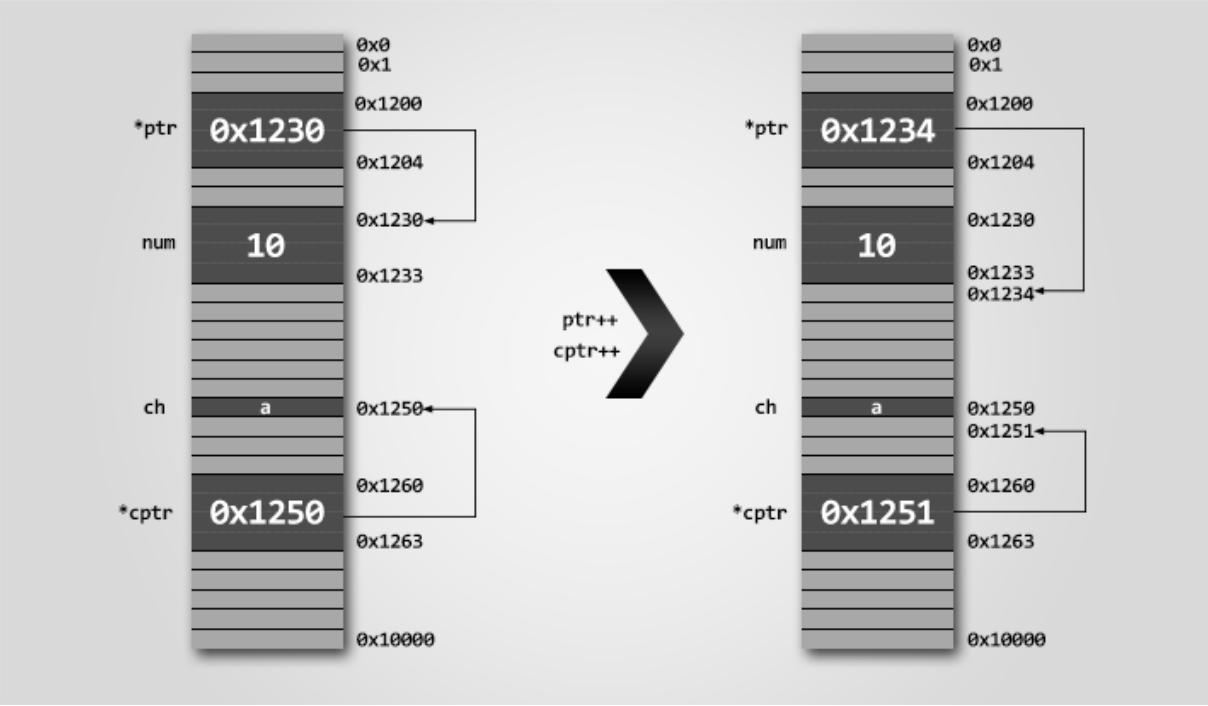
Similarly, decrement operator returns the previous address pointed by the pointer. The returned address is the difference of current pointed address and size of pointer data type.

For example, consider the below statements.

```
int num = 5;    // Suppose address of num = 0x1230
int *ptr;       // Pointer variable

ptr = &num;     // ptr points to 0x1230 or ptr points to num
ptr++;          // ptr now points to 0x1234, since integer size is 4 by
ptr--;          // ptr now points to 0x1230
```

> ***Note:*** *Increment operation increments pointer address by the size of pointer data type.*
> *If an integer pointer* `ptr` *pointing at* `0x1230` *, after* `ptr++` *it will point at* `0x1234` *(assuming integer size is 4 bytes).*
> *If a character pointer* `cptr` *pointing at* `0x1250` *, after* `cptr++` *it will point at* `0x1251` *(since character occupies 1 byte).*



Pointer increment decrement operation memory representation

Example program to perform pointer increment and decrement

Array in memory are stored sequentially, hence is the best example to demonstrate pointer increment, decrement operations.

```c
#include <stdio.h>
#define SIZE 5

int main()
{
    int arr[SIZE] = {10, 20, 30, 40, 50};
    int *ptr;
    int count;

    ptr = &arr[0]; // ptr points to arr[0]

    count = 0;

    printf("Accessing array elements using pointer \n");
    while(count < SIZE)
    {
        printf("arr[%d] = %d \n", count, *ptr);

        // Move pointer to next array element
        ptr++;

        count++;
    }

    return 0;
}
```

Output -

```
arr[0] = 10
arr[1] = 20
arr[2] = 30
arr[3] = 40
arr[4] = 50
```

Pointer addition and subtraction

Pointer increment operation increments pointer by one. Causing it to point to a memory location skipping `N` bytes (where `N` is size of pointer data type).

We know that increment operation is equivalent to addition by one. Suppose an integer pointer `int * ptr` . Now, `ptr++` is equivalent to `ptr = ptr + 1` . Similarly, you can add or subtract any integer value to a pointer.

Adding `K` to a pointer causes it to point to a memory location skipping `K * N` bytes. Where `K` is a constant integer and `N` is size of pointer data type.

Related searches

Programming Language >

C++ Programming Language >

C Programming Language >

C++ Language >

C++ Programming >

Let us revise the above program to print array using pointer.

Example program to demonstrate pointer addition and subtraction

```c
#include <stdio.h>
#define SIZE 5

int main()
{
    int arr[SIZE] = {10, 20, 30, 40, 50};
    int *ptr;
    int count;

```

```
    ptr = &arr[0]; // ptr points to arr[0]

    count = 0;

    printf("Accessing array elements using pointer \n");
    while(count < SIZE)
    {
        printf("arr[%d] = %d \n", count, *(ptr + count));

        count++;
    }

    return 0;
}
```

- When `count = 0`, `(ptr + count)` is equivalent to `(ptr + 0)` which points to `arr[0]` and hence prints 10.
- When `count = 1`, `(ptr + count)` is equivalent to `(ptr + 1)` which points to `arr[1]` and hence prints 20.
- Similarly when `count = 4`, `(ptr + count)` is equivalent to `(ptr + 4)` which points to `arr[4]` and hence prints 50.

Output of above program is same as first program.

Pointer comparison

In C, you can compare two pointers using relational operator. You can perform six different type of pointer comparison `<`, `>`, `<=`, `>=`, `==` and `!=`.

> ***Note:*** *Pointer comparison compares two pointer addresses to which they point to, instead of comparing their values.*

Pointer comparisons are less used when compared to pointer arithmetic. However, I frequently use pointer comparison when dealing with arrays.

Pointer comparisons are useful,

- If you want to check if two pointer points to same location. For example,

```c
int main()
{
    int num = 10;
    int *ptr1 = &num;    // ptr1 points to num
    int *ptr2 = &num;    // ptr2 also points to num

    if(ptr1 == ptr2)
    {
        // Both pointers points to same memory location
        // Do some task
    }

    return 0;
}
```

- If you want to check if a pointer points within an array range. For example,

```c
int main()
{
    int arr[5] = {10, 20, 30, 40, 50};
    int *ptr = &arr[0];  // ptr points to arr[0]

    while(ptr <= &arr[4])
    {
```

```
        // ptr will always point within the array
        // Do some task

        // Move ptr to next array element
        ptr++;
    }

    return 0;
}
```

Example program to demonstrate pointer comparison

Let us re-write our array program without using `count` variable.

```c
#include <stdio.h>
#define SIZE 5

int main()
{
    int arr[SIZE] = {10, 20, 30, 40, 50};
    int *ptr = &arr[0]; // ptr points to arr[0]

    printf("Accessing array elements using pointer \n");
    while(ptr < &arr[SIZE])
    {
        printf("%d \n", *ptr);

        // Move to next array element
        ptr++;
    }

    return 0;
}
```

Rules for performing pointer arithmetic

Pointer arithmetic can be a nightmare if not used correctly. Incorrect pointer arithmetic will lead to you compilation error as well as program crash.

Following are some rules that you must mind while performing pointer arithmetic.

- Result of two pointer addition or subtraction is an integer. For example,

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

int *ptr1 = &arr[0];
int *ptr2 = &arr[4];

int *ptr3 = ptr2 - ptr1; // ERROR -> (ptr2 - ptr1) evaluates to i
```

- Result of pointer and integer addition or subtraction is a pointer. For example,

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

int *ptr = &arr[0];

ptr = (ptr + 2);  // ptr will now point to arr[2]
```

- You must not use multiplication and division operator with pointers.

Valid and invalid examples of pointer arithmetic

```
1
2  int num=10, k=2;                 // Integer variable
3  int *ptr1, *ptr2, *ptr3;         // Integer pointers
4
5  ptr1 = ptr1 - 2;                 // Valid
6  ptr1 = ptr1 - k;                 // Valid
7
8  ptr3 = ptr2 - ptr1;              // Invalid, non-portable pointe
9                                   // Missing cast. See rule 1
```

```
                                    // Missing cast. See rule 1.
ptr3 = (int *) (ptr2 - ptr1)        // Valid

ptr3 = ptr2 - ptr1 - k;             // Invalid, non-portable pointer
                                    // Missing cast. See rule 1.
ptr3 = (int *)(ptr2 - ptr1) + k;    // Valid

ptr1 = ptr1 + 2;                    // Valid
ptr1 = ptr1 + k;                    // Valid

ptr3 = ptr1 + ptr2;                 // Invalid, non-portable pointer
                                    // Missing cast. See rule 1.
ptr3 = (int *) (ptr1 + ptr2)        // Valid

ptr3 = ptr1 + ptr2 + k;             // Invalid, non-portable pointer
                                    // Missing cast. See rule 1.
ptr3 = (int *)(ptr1 + ptr2) + k;    // Valid

ptr1 = ptr1 * 2;                    // Invalid, illegal use of point
ptr1 = ptr1 * k;                    // Invalid, illegal use of point
ptr3 = ptr2 * ptr1;                 // Invalid, illegal use of point
ptr1 = ptr1 / 2;                    // Invalid, illegal use of point
ptr1 = ptr1 / k;                    // Invalid, illegal use of point
ptr3 = ptr2 / ptr1;                 // Invalid, illegal use of point
```


About Pankaj

Pankaj Prakash is the founder, editor and blogger at Codeforwin. He loves to learn new techs and write programming articles especially for beginners. He works at Vasudhaika Software Sols. as a Software Design Engineer and manages Codeforwin. In short Pankaj is Web developer, Blogger, Learner, Tech and Music lover.

Follow on: Twitter | Google | Website or View all posts by Pankaj

*Have a **doubt**, write here. I will **help my best**.*

Before commenting you must escape your source code before commenting. Paste your source code inside
<pre><code> ----Your Source Code---- </code></pre>