Pointers in C · Computer Programmers · Tips and Hacks · C++ (programming language) · C (programming language) · Computer Programming · Survey Question

In C or C++, what are your favorite pointer tricks?

20 Answers

Anders Kaseorg, MIT PhD student in CS; Cofounder of Ksplice, Inc.
Updated Nov 5 2013 · Upvoted by Raghavan Santhanam, Passionate C programmer since 2000. and Ian Hoffman, Computer Science major at Cornell University - I do a lot of personal projects · Author has **555** answers and **7.7m** answer views

I'll start with an important warning about **aliasing**, then cover my three favorite pointer tricks, which are the **double pointer** trick for linked list traversal, finding the **length of an array** without **sizeof** and division, and the Linux kernel's **container_of macro**.

Strict aliasing rules
Before you start using pointer tricks, you should be sure to understand the language's aliasing rules. From the C99 standard §6.5, Expressions:

> 7. An object shall have its stored value accessed only by an lvalue expression that has one of the following types:
> - a type compatible with the effective type of the object,
> - a qualified version of a type compatible with the effective type of the object,
> - a type that is the signed or unsigned type corresponding to the effective type of the object,
> - a type that is the signed or unsigned type corresponding to a qualified version of the effective type of the object,
> - an aggregate or union type that includes one of the aforementioned types among its members (including, recursively, a member of a subaggregate or contained union), or
> - a character type.

Historically, you could get away with violating these rules, and programmers would use pointer casting to access representations of the same data using different types. But modern optimizing compilers will often break that kind of code, because the rules allow them to assume that pointers of different types do not point to the same region of memory!

See the GCC documentation and Understanding Strict Aliasing.

Linked list manipulation with double pointers
Consider the problem of inserting a number into a sorted linked singly-linked list.

```
struct node {
    int data;
    struct node *next;
} *head;
```

A typical solution has two pointers and a few special cases to get right:

Related Questions

- What are some hidden tricks for C/C++ pointers?
- C++ (programming language): What are some well known C++ tricks?
- What are some tricks used in C++?
- What are some pointer hacks in embedded C?
- Are there any tricks for C++?
- Can you solve this C++ (or C) pointer puzzle?
- Why are pointers used in C/C++?
- What are some cool C tricks?
- Where can pointers be placed in C++?
- How can we use pointers in C#?

```c
void insert(struct node *new_node)
{
    struct node *node = head, *prev = NULL;
    while (node != NULL && node->data < new_node->data) {
        prev = node;
        node = node->next;
    }
    new_node->next = node;
    if (prev == NULL)
        head = new_node;
    else
        prev->next = new_node;
}
```

But with clever use of a **pointer to a pointer**, you can do this with no extra cases:

```c
void insert(struct node *new_node)
{
    struct node **link = &head;
    while (*link != NULL && (*link)->data < new_node->data)
        link = &(*link)->next;
    new_node->next = *link;
    *link = new_node;
}
```

The same trick applies to node deletion. It's also useful to store the `previous` links this way in non-circular doubly-linked list implementations, such as the Linux kernel's `hlist` (types, operations).

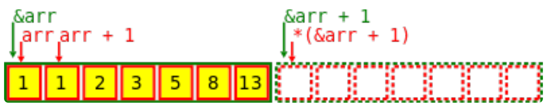
(Of course, in C++, you don't need to write your own linked lists, since the STL provides `std::list` and `std::slist`.)

How long is that array?

It's typical to see loops like

```c
int arr[] = {1, 1, 2, 3, 5, 8, 13};
for (int i = 0; i < sizeof(arr)/sizeof(arr[0]); i++)
    printf("%d\n", arr[i]);
```

That **sizeof** nonsense is ugly, and it's often hidden behind a macro. But we can simplify it using pointer arithmetic.



1. `&arr` is a pointer to the array. It points at the same memory address as `arr`—which decays to a pointer to its first element—but with a different type, whose size is that of the entire array rather than of just one element.
2. That means that `&arr + 1` points at the address after the end of the array.
3. Dereferencing to `*(&arr + 1)` gives the address after the end of the last element (which is again the same address with a different type).
4. Finally, we can subtract the pointer to the first element to get the length of the array: `*(&arr + 1) - arr`.
5. The equivalence between array indexing and pointer arithmetic simplifies this to `(&arr)[1] - arr == 1[&arr] - arr`, saving a pair of parens. This might be that equivalence's only practical use outside the International Obfuscated C Code Contest.

```c
for (int i = 0; i < 1[&arr] - arr; i++)
    printf("%d\n", arr[i]);
```

Having done that, it’s even simpler to make the iteration variable a pointer too:

```
for (int *p = arr; p < 1[&arr]; p++)
    printf("%d\n", *p);
```

(C++11 provides the wonderful new for (int n : arr) syntax for this.)

The container_of macro

The Linux kernel source includes this useful macro that’s invaluable for writing object-oriented code in C. I’ve simplified its definition to remove GCC-specific extensions:

```
#define container_of(ptr, type, member) \
    (type *)((char *)(ptr) - offsetof(type, member))
```

It’s used as follows (example taken from linux/Documentation/kobject.txt). Imagine you have a structure containing another structure:

```
struct uio_map {
    struct kobject kobj;
    struct uio_mem *mem;
};
```

Then, given a struct kobject that you know is a member of a struct uio_map, you might be tempted to get the outer struct uio_map just by casting the pointer:

```
struct kobject *kp = …;
struct uio_map *u_map = (struct uio_map *)kp;
```

But that leads to fragile code that would be broken by the addition of another member to the beginning of struct uio_map. Instead, write:

```
struct uio_map *u_map = container_of(kp, struct uio_map, kobj);
```

37.8k Views · View 456 Upvoters

Upvote · 456 Share

Related Questions More Answers Below

What are some hidden tricks for C/C++ pointers?

C++ (programming language): What are some well known C++ tricks?

What are some tricks used in C++?

What are some pointer hacks in embedded C?

Are there any tricks for C++?

Related Questions

Can you solve this C++ (or C) pointer puzzle?

Why are pointers used in C/C++?

What are some cool C tricks?

Where can pointers be placed in C++?

How can we use pointers in C#?

How do pointers work in C#?

What are some C++ logical tricks?

Does C++ have more advantages of pointers than C?

What does dereference pointer mean in C++?

C++: When to use Reference and Pointer?

How do character Pointers work in C?

What simple C++ tricks should everyone know?

How can I feel comfortable with pointers and function pointers in C/C++ programming?

What is your favorite C# code?

What is your favorite feature of C++?