

Re: Simple question on Pointers

Source: <http://www.tech-archive.net/Archive/VC/microsoft.public.vc.language/2008-11/msg00708.html>

- *From:* "Doug Harrison [MVP]" <dsh@xxxxxxx>
 - *Date:* Tue, 25 Nov 2008 14:25:44 -0600
-

On Tue, 25 Nov 2008 06:44:03 +0700, "Alan Carre" <alan@xxxxxxxxxxxxxxxxxxxx> wrote:

"Robby" <Robby@xxxxxxxxxxxxxxxxxxxxxxxxxxxx> wrote in message <news:C1962AB0-0055-42E0-B8C8-964B666C7462@xxxxxxxxxxxxxxxxxxxx>

In a similar set up, if I do:

```
=====
int main()
{
char a[] = "hello world";
char **ap = &a;

return 0;
}
=====
```

Why does the latter example give the following warning?

```
c:\dts_visual_c++\misc_c_samples\misc_c_samples\utility.c(10) : warning
C4047: 'initializing': 'char **' differs in levels of indirection from
'char
(*)[12]'
```

I know others have responded, but I'm not sure if they've made it perfectly clear... and re-reading this just now I think I will not be much help either!

Though you may find it amusing at least to consider the following:

It is in fact not entirely incorrect to think of &a as being a char**.

Actually, it is entirely incorrect. The type of &a is char(*)[12], i.e. "pointer to array of 12 char", which is very different than char**, or

Re: Simple question on Pointers

"pointer to pointer to char".

After all, 'a' is a kind-of pointer or address to some characters.

No, it's the name of an array. Thinking of it as a "kind-of-pointer" is just going to get you into trouble, because it's not a pointer at all.

Just try writing this down:

```
char a[] = "hello world";  
char* pc = a;
```

No compile warnings. So a is a char* right? WRONG!

The reason it works is that the array "a" undergoes the array-to-pointer conversion in this context, which produces a pointer to its first element. The variable pc is initialized to a copy of this pointer.

Well, sticking to 32 bit pointers, we can cast &a to char** (pointer to char*), and it WILL work (by coincidence really). I say it will, and it does work, strangely, "provided that we do not attempt to reference 'a' "!

Why? Well, certainly the address of a (&a) is a "pointer type" like any address, but 'a' itself is not a pointer, it is a "reference".

No, it's not a "reference", which is a term that has a well-defined meaning in C++. The identifier "a" is simply the name of an array.

What that means is that 'a' merely "represents", or "is an alias for" a memory address where a char* is located (with data {"hello world\0"}).

No, there is no char* in "a". The name "a" is an lvalue for a char[12].

But the address where 'a' is located does not exist. Ok well, it does exist in a sense, but it's more like a "dummy label" at best (and moreover, it is never used). For instance try compiling this:

```
char a[] = "hello world";  
char c = (&a+1)[0][0];
```

Re: Simple question on Pointers

warning C4700: uninitialized local variable 'a' used

What do you mean "uninitialized???" I said `a[] = "hello world"; !`

That's right, 'a' is not initialized, 'a[]' is initialized though.

I would not say that "a[]" is initialized; I would just say that "a" is initialized. There's only one thing that can be initialized here, and it's unusual IME to call it "a[]".

The real problem is that dereferencing `&a+1` is undefined. If we assume that the storage for "a" is at location 0, `&a` will produce a pointer that points to location 0, but it has the type `char(*)[12]`. Therefore, when you say `&a+1`, you are referring to memory location 12, and when you dereference this expression, you're reading past the end of the array "a" into uninitialized memory. Perhaps this is what the warning is trying to say, but it's very misleading for it to describe "a" as "uninitialized", because it certainly is initialized.

So we can certainly write down `char c = (&a)[0][0]` but only because we're extracting `*(&a[0])` and:

The first expression doesn't group like the second. Although the end result is the same, they go through different sequences of evaluations involving different types and are not equivalent in that sense, but they are both equivalent in result to the simpler `a[0]`.

`a[0]`, `a[1]`, `a[12]` are all initialized variables (12 variables). 'a', however, is not.

Again, the array "a" is certainly initialized, and `a[i]` are not separate variables; they are all part of "a". BTW, I thought we were talking about a `char[12]`. :)

--

Doug Harrison
Visual C++ MVP

.