

## Re: De-referencing pointer to function-pointer

**Source:** [http://coding.derkeiler.com/Archive/C\\_CPP/comp.lang.c/2004-05/1194.html](http://coding.derkeiler.com/Archive/C_CPP/comp.lang.c/2004-05/1194.html)

---

**From:** Jack Klein ([jackklein\\_at\\_spamcop.net](mailto:jackklein_at_spamcop.net))

**Date:** 05/12/04

Date: Tue, 11 May 2004 22:43:08 -0500

On Tue, 11 May 2004 13:54:48 +0100, "Edd"

<[eddNOSPAMHERE@nunswithguns.net](mailto:eddNOSPAMHERE@nunswithguns.net)> wrote in comp.lang.c:

> *Jack Klein wrote:*

> > *On Tue, 11 May 2004 03:47:55 +0100, "Edd"*

> > *<[eddNOSPAMHERE@nunswithguns.net](mailto:eddNOSPAMHERE@nunswithguns.net)> wrote in comp.lang.c:*

>

> [ 8< --- snip ]

>

> > *int main(void){*

> > *double (\*f)(double);*

> > *ARRAY funcs;*

> > *InitArray(&funcs, sizeof(sin));*

> >

> > *Either your compiler is broken or you are not invoking it as a C*

> > *compiler and making use of some implementation-defined extension. It*

> > *is a constraint violation to apply the sizeof operator to a function*

> > *designator, and requires a diagnostic.*

> >

> > *The expression "sizeof(sin)" has literally no meaning in C. I have no*

> > *idea what value your compiler generates when you apply sizeof to a*

> > *function. Do you?*

> >

> > [snip]

>

> *I see. Indeed I don't understand what my compiler generates under these*

> *circumstances! However my compiler does not complain about this code in the*

> *slightest, even when I turn on all warnings and support strict ANSI C. I'm*

> *using MinGW under win2k with this command line:*

>

> *gcc -Wall -ansi ptrtest.c -o ptrtest.exe*

>

> *I just tried this on my University system with gcc on unix and I got errors.*

> *Is this a problem with my home compiler, do you think -- should it warn me?*

Not just should, but it required to. When a source program violates syntax or a constraint, the C standard requires the compiler to issue a diagnostic, although it does not specify the format of the

diagnostic.

I haven't used GCC ports much, nor recently, but I think you might need to add `-pedantic`.

```
> >> This program crashes when I run it. Am I doing something undefined
> >> here? I can't see what's going wrong. I think it may be my
> >> understanding of function-pointer syntax is a little lacking, but
> >> what I've got still seems fine to me.
> >
> > Yes, you are doing something undefined here. Function and array names
> > are not converted to pointers when used as operands of the sizeof
> > operator.
> >
> > sizeof(array_name) yields the size, in bytes, of an array, not of a
> > pointer to the element type of the array.
> >
> > sizeof(function_name) would request the compiler to yield the size, in
> > bytes, of the function, not of a pointer to the function. But
> > functions have no sizes accessible to a C program, and that use of
> > sizeof is specifically illegal under the C standard.
>
> I see. Thanks for the clarification.
> This leads me on to the obvious follow-up question -- is there a way of
> achieving the desired result? I can use the alternative method below (which
> works correctly), but it's not quite as elegant as I would like:
>
> int main(void){
> double (*f)(double);
> void *vptr;
> ARRAY funcs;
> InitArray(&funcs, sizeof(void*));
```

No, no, no, no. There is no correspondence between pointers to object types and pointers to functions in C. Even attempting to convert between a function pointer and a pointer to void, in either direction, is completely undefined.

Fortunately, you don't need to. You already have a perfect operand here, just replace the line above with:

```
InitArray(&funcs, sizeof f);
```

`f` is already a pointer to function, not the name of a function, so applying `sizeof` to it is just fine and dandy. Also, since `f` is an object and not a type, the parentheses are not necessary, but harmless if you prefer them.

```
> /* Add some functions to the funcs ARRAY */
> vptr = sin;
> AddElement(&funcs, &vptr);
```

## comp.lang.c: Re: De-referencing pointer to function-pointer

```
> vptr = tan;
> AddElement(&funcs, &vptr);
> vptr = exp;
> AddElement(&funcs, &vptr);
> vptr = log;
> AddElement(&funcs, &vptr);
```

Leave out vptr completely, just omit it from your program. Now that you have uses sizeof f to initialize your structure, you can just do:

```
    AddElement(&funcs, sin);
    AddElement(&funcs, tan);
```

...and so on.

No problem with using the name of a function without () as an argument passed to another function. Unlike with the sizeof operator, this is well defined and automatically converts the name of the function to a pointer to the function.

```
> /* Get the ARRAY element at index 2 */
> f = (double (*)(double))*(void**)GetElement(&funcs, 2);
>
> /* This should now display "f(1.0) = 2.718..."? */
> printf("f(%lf) = %lf\n", 1.0, f(1.0));
>
> return 0;
> }
>
> Thanks for you reply,
> Edd
```

I have copied this from your original post:

```
> /* Get the address of the kth element */
> void *GetElement(ARRAY *a, unsigned k){
> return (char *)(a->base) + (k * a->elsz);
> }
```

The first thing I would do is change the return type to "const void \*", but that's not mandatory.

To retrieve a function pointer from your array, you can get rid of all that almost indecipherable casting by doing this:

```
void *vp;
vp = GetElement(&funct, 2);
memcpy(&f, vp, sizeof f);
```

The latter two lines can be combined, with rather less readability, to eliminate the need for the pointer to void:

comp.lang.c: Re: De-referencing pointer to function-pointer

```
memcpy(&f, GetElement(&func, 2), sizeof f);
```

All the nasty casts are gone!

--

Jack Klein

Home: <http://JK-Technology.Com>

FAQs for

comp.lang.c <http://www.eskimo.com/~scs/C-faq/top.html>

comp.lang.c++ <http://www.parashift.com/c++-faq-lite/>

alt.comp.lang.learn.c-c++

<http://www.contrib.andrew.cmu.edu/~ajo/docs/FAQ-acllc.html>