

Re: pesky Pointers !!

Source: http://coding.derkeiler.com/Archive/C_CPP/alt.comp.lang.learn.c-cpp/2004-12/0712.html

From: Edd (*edd_at_NOSPAMHEREununswithguns.net*)

Date: 12/21/04

Date: Tue, 21 Dec 2004 01:28:27 +0000

Rich wrote:

```
>
>> '*pTest' is type 'string'
>> Your function needs an argument of type 'string*' ('pointer to string').
>> 'pTest' is type 'string*', and contains the address of 'sTest'.
>>
>> Write:
>>
>> StringTest(pTest);
>>
> Hi Thanks that worked, but I still dont understand why it worked, sorry :(
>
> pTest looks just like a normal variable, which I thought was what a a
> reference was used for ???
```

References are not pointers, but in many ways they are similar. You may come on to references soon in your course. If you've already covered them, I'd go back and read over your material on them.

```
> A reference as a parameter in a definition allows the programmer to pass
> a reference without really knowing, i.e. no need to do &x or *x just x
> and the function takes it as a reference instead of a copy.
```

If the programmer doesn't need to know that 'call-by-reference' is being used, a function often takes a const reference e.g:

```
//prototype of OP's function using const reference.
void StringTest( const string& s );
```

This is an agreement between the function and the caller that the function may access the string passed directly, but may not modify it. i.e. if you went a stage further and reconstructed your function to edit the string, your compiler should give a diagnostic.

Similar things apply to a function with this prototype:

```
void StringTest( const string* s );
```

alt.comp.lang.learn.c-c++: Re: pesky Pointers !!

However, if the const was removed in the above examples, the function would be free to modify the contents of the string referenced/pointed-to. In this case, functions are often given a name that indicates the argument will be modified on call e.g:

```
void Increment(int &i) { ++i; }
```

Basically, if a function is to modify an object, that should be made obvious to anyone wanting to call said function. Else a pointer or reference to a constant object should really be used in the function definition, as a kind of unwritten contract.

```
>
> From what I have learnt about pointers and syntax using pointers it is
> fairly straightforward you read from right to left e.g
>
> int* px // px is a pointer to an int
>
> *px dereferences the pointer to get the value
  ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
```

Remember what you've said here for later...

```
>
> &px gives the address that *px points to
>
> hopefully I have the above right?
>
> So then if a function expects a parameter string* s
>
> This is saying I need a pointer to a string passed to me??
>
> So I thought I needed to pass the pointer as an argument thus requiring
> either the use of the * or & before the pointer to show that it is a
> pointer I am passing.
```

If pTest is a pointer-to-string, *pTest is the string it points to (remember, from above?). Your function expects a pointer to a string. By calling

```
StringTest( *pTest );
```

you are passing a string to a function which expects a pointer to a string. You don't want to use * to dereference pTest and so you should just do:

```
StringTest( pTest );
```

because that passes a pointer-to-string to the function, which is exactly what it expects.

>
> *by passing pTest just like that it looks like a normal variable name ??*
>
> *e.g if I just called the pointer Test then it would be confusing.*
>
> *Now I am completely confused again doh!!*

I hope I've helped to un-confuse you to some extent.

> *BTW*
>
> *I notice a couple of different C++ styles one which our tutor insists we*
> *use but I dislike*
>
> *she insists on this type of style for parenthesis*
>
> *a [10] // array*
> *func (10, 20) // function call*
>
> *I have also seen*
> *a[10]*
> *func(10,20)*
>
> *I prefer an in between like*
> *a[10]*
> *func(10 ,20)*
>
> *I use this because having the parenthesis next to the variable or*
> *function name makes it look like it is part of it rather having the*
> *variable part then a space then parenthesis looks kind of odd to me, I*
> *can understand the spaces after first parenthesis and before last*
> *parenthesis but why do some people insist on having a space before the*
> *first parenthesis as well???*
>
> *I know style is a matter of preference, but sometimes I think too many*
> *spaces makes it more difficult to read than no spaces.*
>
> *I try to voice this with our tutor but she insists on the all spaces*
> *style, this style I know is popular with Java programmers, but when did*
> *it become a style in C++??*

My two cents about style: if your marks will be affected by not using your teachers style then use her style. This really, really should not be the case imho, as anyone who feels they are able to teach C++ should be able to read code of any sensible style. Otherwise use what you're comfortable with so long as it's sensible and used in a consistent manner.

I find it usually only takes a couple of minutes to become accustomed to someone else's style. I'd be surprised if your teacher is any different.

Edd

Re: pesky Pointers !!

alt.comp.lang.learn.c-c++: Re: pesky Pointers !!

--

Edd Dawson	www.nunswithguns.net		"Arthur! My moustache is
To email me, cut the crap.			touching my brain!"
edd@nunswithcrapguns.net			- The Tick