What is a long pointer?

Asked 14 years, 3 months ago Modified 8 months ago Viewed 30k times

47

I am reading a book and it mentions certain data type as being long pointer. Just curious about what that meant. Thanks.

c++ c pointers

Share Follow

edited Feb 23, 2010 at 0:33
Peter Alexander
53.9k ● 14 ● 120 ● 169

asked Feb 23, 2010 at 0:23
numerical25
10.7k ● 36 ● 135 ● 214

3 Historical explanation from MSDN: msdn.microsoft.com/en-us/library/windows/desktop/...
– Nate Glenn Aug 31, 2014 at 12:01

2 Answers

Sorted by: Highest score (default)

51

Some processors have two types of pointers, a near pointer and a far pointer. The near pointer is narrower (thus has a limited range) than a far pointer. A far pointer may also be a long pointer.

Some processors offer relative addressing for things nearby. A long pointer may indicate that the item is not close by and relative addressing cannot be used.

In any case, long pointers are a platform specific issue and may not be portable to other OSes or platforms.

Edit: *(further explanation and usage of relative addressing)*

Address distances are less of a high level concept and more of an assembly language concept. The **distance** is measured from the program counter (either the current address or the next address) and the start of the object (function or data). If the location is greater than the limit for a small, relative pointer, a longer pointer will be necessary.

Example: Given a system with 32 bit "long" addressing and 8-bit relative addressing. The relative distance would allow for at least 127 bytes in the forward (positive value) or previous (negative) direction. If the target is 1024 bytes away, a full 32-bit pointer must be used.

This is an optimization feature based on the concept that most instructions and data are nearby. The majority of loops have a small distance between the start of the loop and end of the loop. These employ relative addressing for execution.

Most data is nearby, whether a data constant or a variable. In more detail, the data is near a *frame* or reference point. Local variables are placed on the stack, relative to a frame or base address. This base address is the start of the stack *before the function is executed*. Thus the data can be accessed using addressing relative to the stack frame start.

The processors allow compilers to use specialized instructions for relative (near) addressing. On many processors, the instructions to use relative addressing are smaller than instructions using long addressing. Thus the processor requires less fetching from the instruction cache and the instruction cache can hold more instructions.

Long and short, near and far, addressing may depend on the *scope* of the data or function. There are other factors involved, such as position-independent code (PIC), virtual memory and paging.

Share Follow

edited Sep 22, 2023 at 15:45
Tom Tom Lee
366 ● 1 ● 3 ● 11

answered Feb 23, 2010 at 0:27
Thomas Matthews
57.3k ● 17 ● 103 ● 158

Is this distance measured by it's scope ? How many times it's encapsulated it ? Thanks! – numerical25 Feb 23, 2010 at 0:35

@numerical25: Please explain "How many times it's encapsulated it?". Addressing isn't encapsulated. – Thomas Matthews Feb 23, 2010 at 1:16

2 "scope" does not exist any more at this level - only a long array of bytes. All your code gets translated into assembly, and that is laid out in memory. When things are close enough together the compiler generates near pointers, when further away the compiler needs to generate far pointers. – Justin Smith Feb 23, 2010 at 3:22

If you downvote this answer, please leave an explanation. Thanks. – Thomas Matthews Jan 5, 2020 at 19:11

38

Depending on how old the book is, it might be referring to segmented architectures, where there were two different "sizes" of pointer: near pointers, which pointed into the local segment (and could be fitted into 16 bits), and far or long pointers, which could point into another segment (and therefore were bigger). This is why you see types like LPVOID in the Win32 API: LPVOID is a "long (far) pointer to void", i.e. a pointer to anywhere in memory.

The use of LP and NP is a hangover from Win16 and the segmented processor architectures of the time. In modern Windows, with its flat virtual address spaces, near and far pointers are generally of archaeological interest only: there is only one kind of pointer and you can ignore the "near" and "long/far" qualifiers.

Share Follow

answered Feb 23, 2010 at 0:29
itowlson
74.4k ● 17 ● 163 ● 159

1 the book is using Direct X 9, MSVS C++ 2008 express. copyright 2010. And some of the datatypes begin with LP. But I am getting 2 different answers so I am guessing they can mean 2 different things

– numerical25 Feb 23, 2010 at 0:33

Maybe you could post one of the mentions in context (e.g. the sentence in which it appears) -- we could then explain more specifically the significance in that context. E.g. is it just a side note explaining why some of the data types have the LP prefix? – itowlson Feb 23, 2010 at 0:49

1 For a definition of segmentation, I'm going to take the coward's way out and point you at Wikipedia: en.wikipedia.org/wiki/X86_memory_segmentation – itowlson Feb 23, 2010 at 0:52

15 @numerical25: the reason that `LP` is still used in the names of pointer types (like `LPDWORD`) is just a legacy that holds over from the Win16 days - an `LPDWORD` on Win32 is just a pointer. On Win16 it's a 'long' or 'far' pointer, but that distinction doesn't exist in Win32 anymore. It's still around in the type name so code that was written for Win16 and/or copied over from that era will port easily. That was hugely important back when Win32 was emerging. – Michael Burr Feb 23, 2010 at 1:31

Actually, the concept of having near and far pointers predates Win16 to at least the early 8-bit microprocessors (think TRS-80 time period). By using near pointers, more code could be put into *memory*, and memory was very expensive (limiting the quantity of memory). The Intel processors have different sized instructions, with the most popular instructions have smaller lengths. For example, the Signetics 2650 had Load Relative and Load Absolute instructions using "near" and "far" pointers, respectifully. – Thomas Matthews May 7 at 20:58