What does C expression ((void(*)(void))0)(); mean?


```
((void(*)(void))0)();
```

So we have integer 0 type casting to this tricky type `(void(*))(void)` and then executing it. Source claims that this should work, but what does it actually?

This must be one of those C jokes like `#define TRUE FALSE`, I suppose.

c

asked Dec 3 '13 at 16:56

user2984878

48 4

3 Which source claims that this should work? Do you have a link? It might be easier to understand in context. – anatolyg Dec 3 '13 at 16:58

1 It is casting to a function pointer. Looks like a segfault waiting to happen to me. – jim mcnamara Dec 3 '13 at 16:59

3 Non-standard `abort()`, maybe? ;) – Guido Dec 3 '13 at 17:08

1 *Jekyll*'s answer inspires me to ask you whether your question arose in the context of embedded systems? – alk Dec 3 '13 at 17:14

1 This was mentioned in context of a joke: `if(cmd == RESET) // if command is RESET ((void (*)(void))address)(); // something from black magic` – user2984878 Dec 3 '13 at 17:57

5 Answers

This is a function expecting no arguments and returning no value:

```
void f(void)
```

This is a pointer to a function expecting no arguments and returning no value:

```
void (*p)(void)
```

This is the *type* of that pointer:

```
void (*)(void) /* just remove the p! */
```

This is that type in parentheses:

```
(void (*)(void))
```

This is a cast to that type (the type in parentheses, followed by a value):

```
(void (*)(void))0
```

Still with me? so far we have the integer value 0 cast to a pointer-to-function-that-takes-no-arguments-and-returns-nothing.

The cast is an expression with pointer-to-function type. When you have one of those you can call it like this:

```
(your expression here)(arguments to the function)
```

The first set of parentheses are just for precedence, and sometimes might not be needed (but this time they are). The end result:

```
((void (*)(void))0)(/* no args */);
```

Takes the value 0, casts it to pointer-to-function-expecting-no-arguments-and-returning-nothing, and calls it, supply no arguments.

answered Dec 3 '13 at 17:03
Wumpus Q. Wumbley
8,873 8 25

The syntax to cast address to a function pointer and then call it would look like this:

```
((void (*)(void))address)();
```

It might be clearer to do something like this though:

```
void (*fptr)(void) = (void (*)(void))address;
fptr();
```

Said that `((void(*)(void))0)();` instruction is used to jump to 0 in firmwares usually. It is a bit improper because it actually calls in 0 instead of jumping to 0, but practically it won't make any difference (a fw hot reboot will be performed)

edited Dec 3 '13 at 17:13
alk
36.6k 5 24 69

answered Dec 3 '13 at 17:00
Jekyll
996 6 10

What does "*fw*" mean, please? – alk Dec 3 '13 at 17:10

2 firmware, there is the boot address is usually in 0 (or 0xFFFF0000 for arm or i386), this jump (call) will cause the processor hot restart. If you want more information look for ***jump to reset vector*** – Jekyll Dec 3 '13 at 17:12

+1 can you provide a link that shows similar code? – Shafik Yaghmour Dec 3 '13 at 17:21

@ShafikYaghmour I will look for it, in the meantime you can see a small variant in Arduino at learn.adafruit.com/system/assets/assets/000/010/293/original/... Note that this reset is not very suitable for big controllers (like ARM11 or more recent) in which usually it is fired an internal watchdog in order to achieve a more "secure" HW reset. – Jekyll Dec 3 '13 at 17:44

This treats `NULL` as a function pointer and executes it, it should raise a sigbus or similar on most systems.

```
void(*)(void)   <- type, function pointer taking no arguments and returning no value
(void(*)(void)) <- cast to above type
((...)0)        <- cast NULL/0 to said type
((...)0)()      <- execute the cast value as a function
```

answered Dec 3 '13 at 16:59
Kevin
26.5k 7 39 77

On some embedded system (AVR microcontroller, for example) this might be a way to implement a jump (a call, really) to the reset vector.

You could restart the SW this way, especially if you have disabled interrupts before this.

answered Dec 3 '13 at 17:14
teroi
135 1 9

The AVR's will normally load the user application program at address 0. A bootloader program which is loaded at a much higher address, might jump to (call the function at) address 0. So, the cast and call to the 0 address is a useful construct in this environment.

answered Sep 1 '14 at 1:18
user3995709
11 1
