

# Re: libc/printf bug

---

*Source:* <http://linux.derkeiler.com/Newsgroups/comp.os.linux.development.apps/2006-02/msg00385.html>

---

- *From:* Jan Panteltje <pNaonStpealmtje@xxxxxxxx>
  - *Date:* Mon, 27 Feb 2006 20:28:21 GMT
- 

On a sunny day (27 Feb 2006 12:09:54 -0800) it happened "bill pursell" <bill.pursell@xxxxxxxx> wrote in <1141070994.179527.261050@xx>:

The following code exhibits unexpected behavior. Either it's a bug in libc, or the author (me) is blind as a bat. Am I missing something here, or is something horribly wrong? It appears that printf is mangling the address unless the second argument gets cast. I don't see that the cast should have any effect at all on the output. Any thoughts?

```
[tmp]$ cat printf_bug.c
```

```
int
main()
{

float a=0;

printf("%p\n", &a);
printf("%x %p\n", a, &a);
printf("%x %p\n", (int)a, &a);
}
[tmp]$ ./a.out
0xfeb6764
0 (nil)
0 0xfeb6764
[tmp]$
```

How would you print a float in hex? I think print will convert it to integer, and that is 0.  
`%x', `%X'

Print an integer as an unsigned hexadecimal number. `%x' uses lower-case letters and `%X' uses upper-case. \*Note Integer Conversions::, for details.

So seems printf gets confused because you present it with a float when it wants an integer. This behavior I have seen in other cases where wrong or missing arguments were presented too.

Re: libc/printf bug

Maybe it has to do with stack space where the vars are when the function is called?

.