

## Re: Convert float to double – weird failure

---

*Source:* <http://coding.derkeiler.com/Archive/General/comp.programming/2007-10/msg00398.html>

---

- *From:* Joe Wright <[joewwright@xxxxxxxxxxx](mailto:joewwright@xxxxxxxxxxx)>
  - *Date:* Sun, 28 Oct 2007 21:36:19 -0400
- 

bill robertson wrote:

"tugboat90" <[bmshipe@xxxxxxxxxxx](mailto:bmshipe@xxxxxxxxxxx)> wrote in message  
[news:1193530070.911559.316470@xx](mailto:news:1193530070.911559.316470@xx)

<snips>

Thank you for the responses, but I understand how the numbers are represented inside of a computer. I think I phrased my question wrong.

I don't believe that you do understand how numbers are represented inside a computer. If you did, you would have understood the excellent answers that have been given. You still seem to think the numbers are decimal but they are binary and not an exact binary interpretation of the decimal number. For example, your 1.2 is represented inside the computer you are probably working on is given below and there are many systems on which this isn't correct.

Float 1.2 is uninterpreted hex 0x3f99999a  
float = 1.20000004768371582000000000000000  
sign = 0x0  
mantissa = 0x19999a  
exponent = 0x7f  
bias = 0x7f  
radix = 0x2  
value =  $(-1)^{\text{sign}} * (\text{radix})^{(\text{exponent} - \text{bias})} * (1).\text{mantissa}$   
where calculations are in binary (for radix 2)

When I converted from a float to a double, why did it add random junk at the end of the number instead of making it zeros? If the random junk is there, how do I get rid of it?

Thanks!

It's converting the float representation of 1.2 which is 0x3f99999a to a double representation which is not even 0x3f99999a00000000.

## Re: Convert float to double – weird failure

Here's what a float looks like in Little Endian (ix86)

```
typedef struct {
  unsigned int mantissa:23;
  unsigned int exponent:8;
  unsigned int negative:1;
} ieee_float;
```

Here's what a double looks like in Big and Little Endian (ix86)

```
#if BIGENDIAN
  unsigned int s : 1;
  unsigned int e : 11;
  unsigned int m : 20;
  unsigned int m2 : 32;
#else
  unsigned int m2 : 32; // 2nd part of mantissa
  unsigned int m : 20; // 1st mantissa
  unsigned int e : 11; // exponent – which is why range of double is larger
  unsigned int s : 1; // sign
#endif
```

Hi Bill, let me jump in here please. Constant 1.2 has type double.

```
float f = 1.2;
```

```
00111111 10011001 10011001 10011010
Exp = 127 (1)
00000001
Man = .10011001 10011001 10011010
1.20000005e+00
```

It's been rounded up in this case.

```
double d = f;
```

Assigning float to double changes nothing.

```
00111111 11110011 00110011 00110011 01000000 00000000 00000000 00000000
Exp = 1023 (1)
000 00000001
Man = .10011 00110011 00110011 01000000 00000000 00000000 00000000
1.2000000476837158e+00
```

```
double d = 1.2;
```

```
00111111 11110011 00110011 00110011 00110011 00110011 00110011 00110011
Exp = 1023 (1)
000 00000001
Man = .10011 00110011 00110011 00110011 00110011 00110011 00110011
1.2000000000000000e+00
```

Re: Convert float to double – weird failure

## Re: Convert float to double – weird failure

It makes no sense to represent a 32-bit float value beyond 9 decimal digits, or a 64-bit double beyond 17. I can't imagine how expressing the binary float or double as hex helps at all.

```
float f = 1.2; /* 0x3f99999a */  
double d = 1.2; /* 0x3ff3333333333333 */
```

It's data to be sure, but not information.

The mantissa of floating point types denotes precision of the type. For float it is 24 bits, for double 53 bits. The high order bit of the normalized mantissa being always 1, its position in the representation is taken by the low order bit of the exponent.

--

Joe Wright

"Everything should be made as simple as possible, but not simpler."

---- Albert Einstein ----

.