

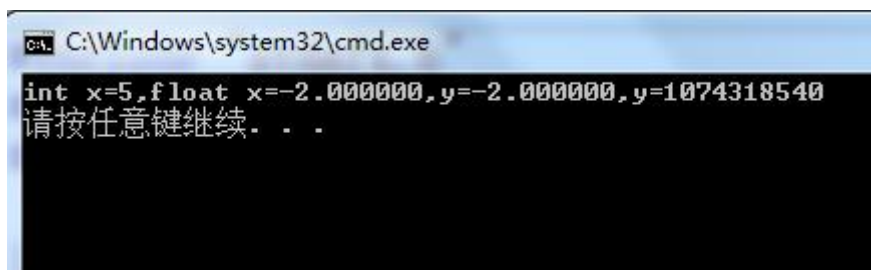
```
//限定x86平台
#include "stdafx.h"
#include <windows.h>
#include <string.h>

int _tmain(int argc, _TCHAR* argv[])
{
    int x=5;
    float y=3.1f;

    printf("int x=%d, float x=%f, y=%f, y=%d\n", x, x, y, y);

    return 0;
}
```

输出：



为什么？

printf 函数按照 cdecl 调用约定传参：

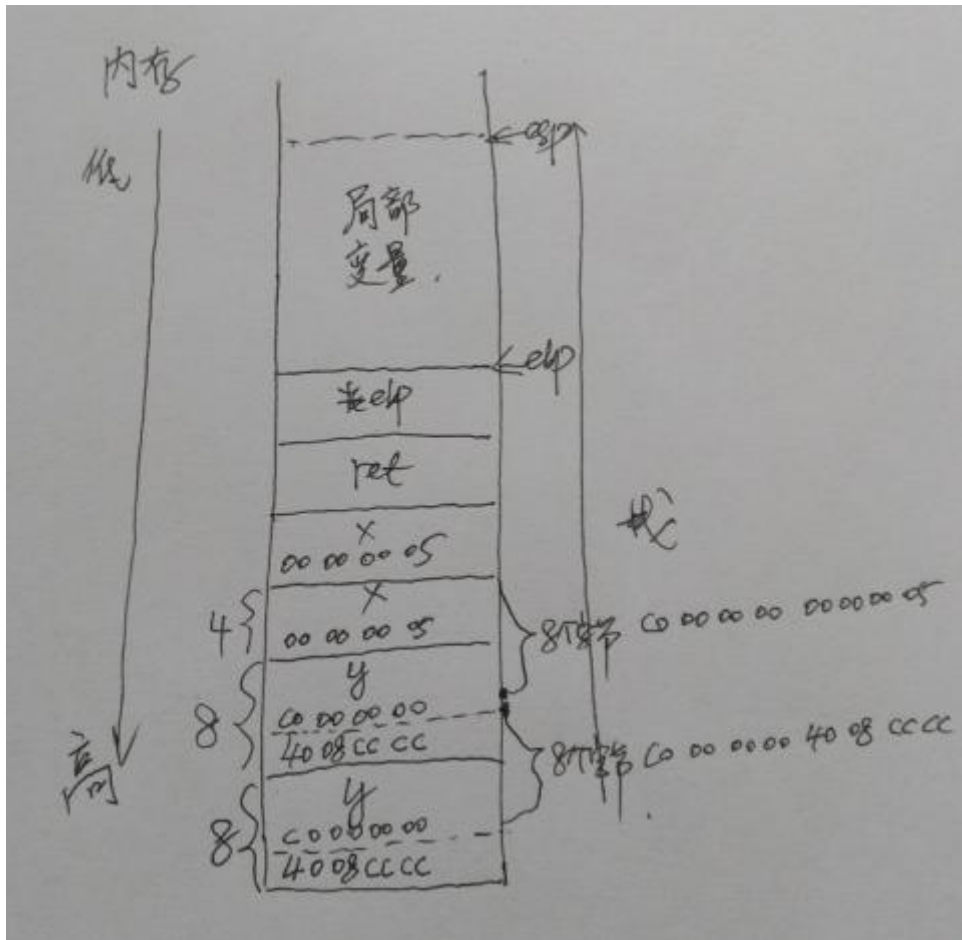
y(3.1f)作为 float 入栈，会提升为 double，因此在栈上，2 次入栈，各占 8 个字节。

然后紧接着是 2 个 x 入栈，分别占 4 个字节。

单精度浮点数 3.1f 的二进制内存为：40466666

3.1f 提升为双精度 8 个字节浮点数后，二进制内存为：4008cccc0000000

如下图所示：



在打印的时候，第一个 x 被当做整数打印，所以从栈上取 4 个字节，打印出来是 5 没有问题。当时打印第二个 x 的时候，被当做 float 类型，会用第二个 x 的 4 个字节，和 y 的低 4 个字节组成一个 8 个字节的 float，打印出来是 -2.0  
打印 y 的时候，由于 y 的低 4 个字节被 x 占用了，它会继续占用第 2 个 y 的低 4 个字节，组成一个新的 y，打印出来的是 -2.0  
这就是你看到的现象：x 和 y

`c000000000000005 c00000004008cccc |`

这就是第 2 个 x 的 4 个字节和第一个 y 的低 4 字节组成的数被当做 x 的 float 来打印（实际上是 x 被当做 float 了之后，占用了它旁边的 y 的低 4 字节）：

`c000000000000005`，这个数，约等于 -2.0。

第一个 y 的高 4 字节和第二个 y 的低 4 字节（第一个 y 的低 4 字节被第 2 个 x 占用了，因此它会继续占用第二个 y 的低字节）组成的数：`c00000004008cccc`，这个数，也约等于 -2.0。如下图所示：红框中的就是新的 x 和 y 的 float 值：

File Edit View VAssistX Project Build Debug Tools Test Window Help

Process: [10000] ping\_demo.e Thread: [8948] Main Thread Stack Frame: ping\_demo.exe!wmain(i

Memory 1

Address: 0x0023FD7C

栈内存布局，第2个x与第1个y的低4个字节组合成了一个float  
第1个y的高4个字节与第2个y的低4个字节组成一个 float，第2个y剩下4个字节

0x0023FD7C	00000005	00000005	c0000000	4008cccc	c0000000	4008cccc	00000000	00000000	7efde000	cccccccc	cccccccc	cccccccc	cccccccc
0x0023FDB8	cccccccc	cccccccc	cccccccc	cccccccc	cccccccc	cccccccc	cccccccc	cccccccc	cccccccc	cccccccc	cccccccc	cccccccc	cccccccc
0x0023FDF4	cccccccc	cccccccc	cccccccc	cccccccc	cccccccc	cccccccc	cccccccc	cccccccc	cccccccc	cccccccc	cccccccc	cccccccc	cccccccc
0x0023FE30	cccccccc	cccccccc	cccccccc	cccccccc	cccccccc	cccccccc	cccccccc	cccccccc	cccccccc	cccccccc	cccccccc	cccccccc	cccccccc
0x0023FE6C	cccccccc	00000005	cccccccc	0023fee8	008b19a8	00000001	00565ed0	00569018	e623297a	00000000	00000000	7efde000	00000000
0x0023FEA8	00240000	00000000	0023fee8	0000001d	0023ff0c	008b107d	e68bbd12	00000000	0023fed0	008b17ef	0023fedc	755a336a	7efde000
0x0023FEE4	7efde000	7453af4f	00000000	00000000	7efde000	00000000	00000000	00000000	0023fee8	00000000	fffffff7	776358e5	032e967f
0x0023FF20	775f98d5	008b1078	7efde000	00000000	00000000	00000000	00000000	008b1078	7efde000	00000000	00000000	00000000	00000000
0x0023FF5C	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000

Disassembly printf.c windef.h ping\_demo.cpp

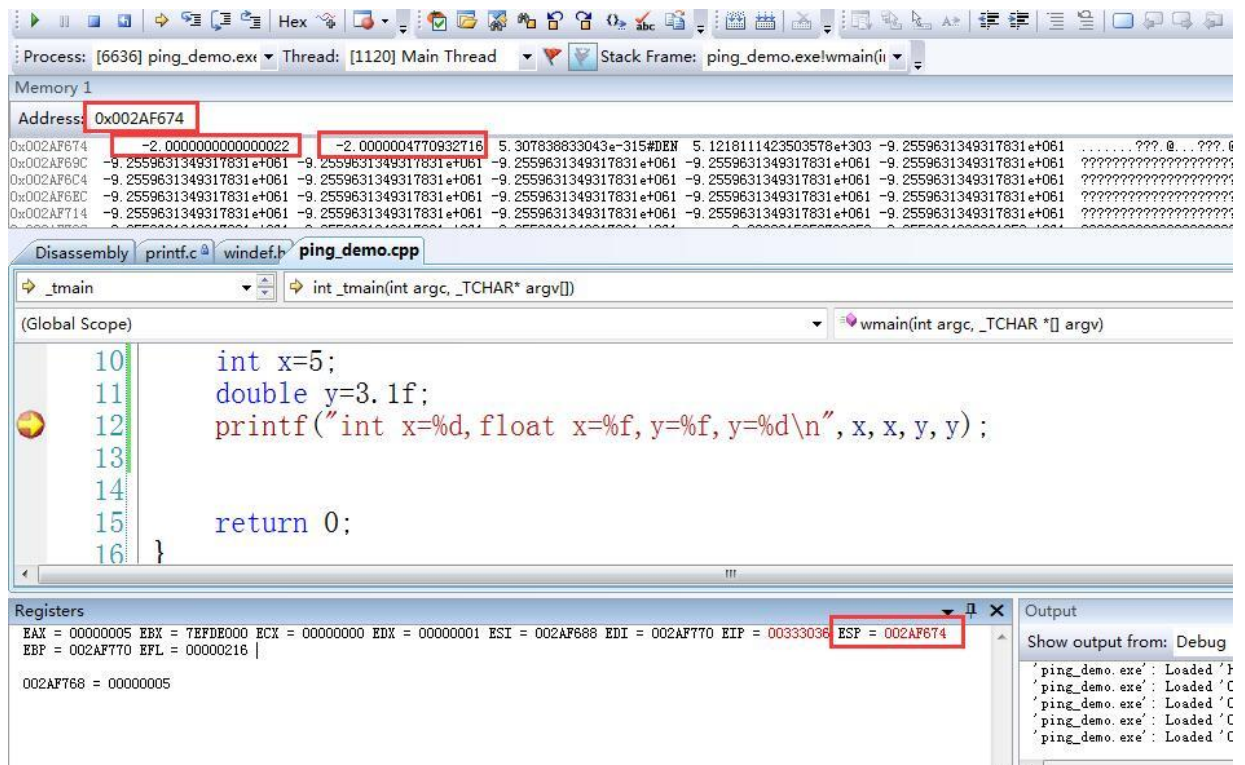
int x=5

```
8 int _tmain(int argc, _TCHAR* argv[])
9 {
10     int x=5;
11     float y=3.1f;
12
13     printf("int x=%d, float x=%f, y=%f, y=%d\n", x, x, y, y);
14
15     return 0;
```

Registers

EAX = 00000005 EBX = 7efde000 ECX = 00000005 EDX = 00000001 ESI = 0023FD94 EDI = 0023FE78 EIP = 008B358A ESP = 0023FD7C  
EBP = 0023FE78 EFL = 00000206

栈顶指针地址



The screenshot shows a debugger window for the process 'ping\_demo.exe' at thread '[1120] Main Thread'. The 'Memory 1' window displays a memory dump starting at address 0x002AF674. Two values are highlighted with red boxes: -2.0000000000000022 and -2.0000004770932716. The 'Disassembly' window shows the source code for 'ping\_demo.cpp' with the following code:

```
10 int x=5;  
11 double y=3.1f;  
12 printf("int x=%d, float x=%f, y=%f, y=%d\n", x, x, y, y);  
13  
14  
15 return 0;  
16 }
```

The 'Registers' window shows the following values:

```
EAX = 00000005 EBX = 7EFDE000 ECX = 00000000 EDX = 00000001 ESI = 002AF688 EDI = 002AF770 EIP = 00333036 ESP = 002AF674  
EBP = 002AF770 EFL = 00000216  
002AF768 = 00000005
```

The 'Output' window shows the following output:

```
'ping_demo.exe': Loaded 'C:\Windows\System32\GDI32.dll'  
'ping_demo.exe': Loaded 'C:\Windows\System32\USER32.dll'  
'ping_demo.exe': Loaded 'C:\Windows\System32\GDI32.dll'  
'ping_demo.exe': Loaded 'C:\Windows\System32\USER32.dll'  
'ping_demo.exe': Loaded 'C:\Windows\System32\GDI32.dll'  
'ping_demo.exe': Loaded 'C:\Windows\System32\USER32.dll'
```

因此，第 2 个 y 只剩 4 个字节了，被当做了 4 个字节的整数打印