

**%o:**

通过循环算出 x 的八进制数，将每一位用数组保存下来，直接使用 `putch` 输出。

**%+:**

设置 `padc` 为 '+'，传入 `printnum` 函数，`printnum` 函数已经实现了这一块，不做别的修改。

**%n:**

先使用 `getuint` 获得传入的地址，其类型为 `uint`，强制赋给 `char*`；判断其是否为空，若为空，输出预设错误信息；判断 `putdat` 所指的 `uint` 数据是否大于 127，若是，则输出错误信息，并将 `char*` 指向的数据设为 -1；若非两者，将 `putdat` 所指的数据赋给 `char*` 指向的数据。

**%-:**

将 `padc` 设为 '-'；在 `printnum` 函数的最后，加上一条：判断若 `padc == '-'` 则打印 `width` 个空格。为了处理递归带来的麻烦，将递归体写为另一个函数，再由原函数调用它。

**Debuginfo\_eip:**

没什么可讲的，依葫芦画瓢。

**Backtrace:**



如图：

- ①eip, arg1,arg2,arg3,arg4,arg5 分别在%ebp+4, %ebp+8, %ebp+12, %ebp+16, %ebp+20, %ebp+24 的位置上。
- ②：将 eip 传给 debuginfo\_eip，获得结构体 info。则文件名、所在行数、函数名、从函数头开始经过的字节数分别为 info.eip\_file, info.eip\_line, info.eip\_fn\_name, 和 eip-info.eip\_fn\_addr。
- ③输出这些信息后，将%ebp 所指向的数据赋给%ebp，继续①，直到%ebp==0。

## Buffer hack：

由于 printf 中%n 可以将当前输出字节数赋给某一个内存地址，我尝试借助这个方法将函数的 return address 修改为 do\_overflow 的地址。

- ①使用 read\_pretaddr() 得到返回地址，记为 retaddr。
- ②将目标地址 do\_overflow+3 的 4 个字节分别记录在四个变量里 a0,a1,a2,a3。
- ③调用四次 cprintf，并保证传入的字符串长度刚好等于 ai，使其赋给相应的 retaddr 第 i 个字节。
- ④将原来的 retaddr 赋给现在%ebp+4 的位置，使其正常返回至 overflow\_me。