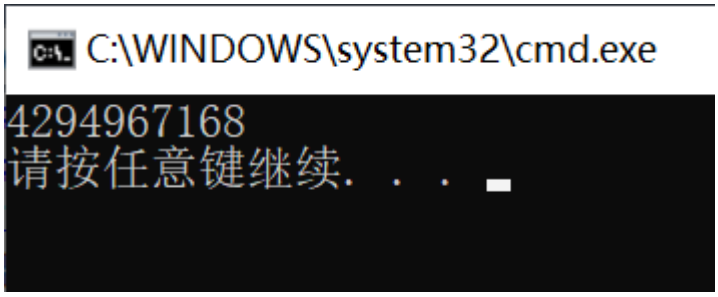
我们知道C语言是一门强类型的语言，那么就要求只有类型相同或者相近时才可以进行赋值，在不同但是类型相近的类型赋值时就会悄悄的发生隐式类型转换。

请看下面这个代码：请问它输出什么？

```
#include <stdio.h>
int main()
{
    char a = -128;
    printf("%u\n", a);
    return 0;
}
```

如果你觉得它输出的是 128 或者是 -128　那么：

下面请看运行结果：



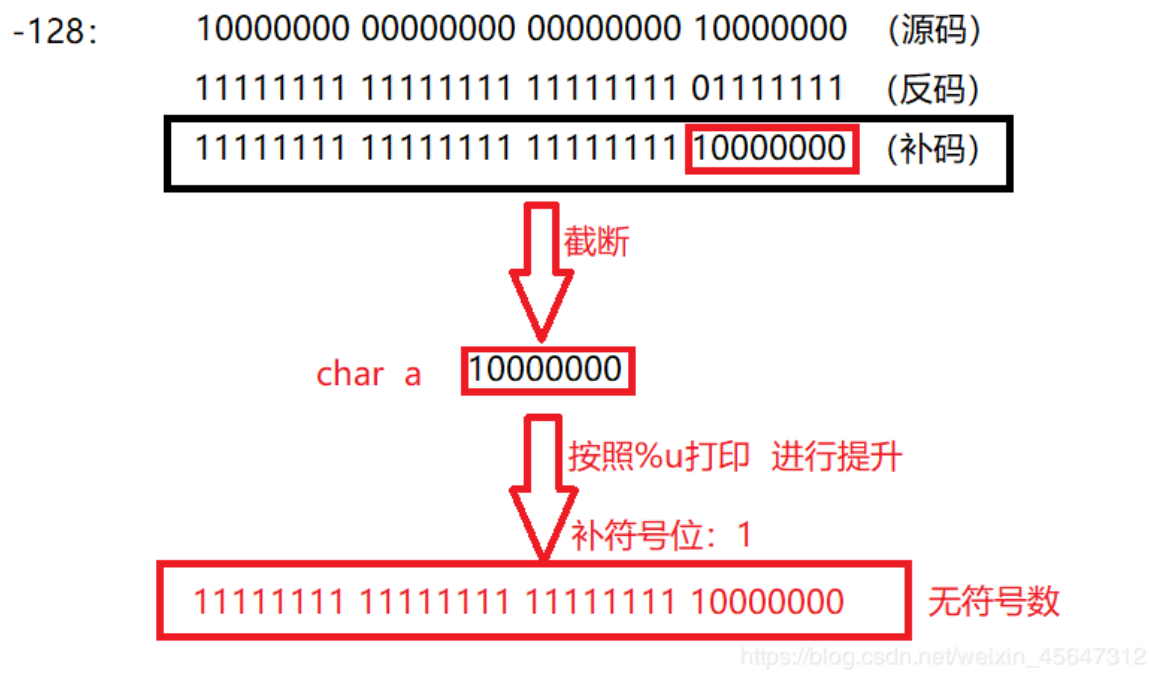
这是为什么呢?

> **这就是因为在这个代码中发生了隐式类型转换**
>
> 1. 当把 -128 这样一个数字要赋给a时，由于 -128 是一个整型占四个字节，而a是一个字符型占1个字节，在赋值要发生 整型的截断
> 2. 在截断发生后，要按照 %u（无符号的整型）格式打印，就要发生整型的提升

> **源码-反码-补码：**
>
> 1. 整型在内存中存储的是补码
> 2. 源码 (符号位不变，其余位取反) --->反码 （反码+1） ---> 补码
> 3. 正数的源码，补码相同

> **整型的截断：**
>
> 将二进制位进行截断（截补码）
>
> **整型的提升：**
>
> 在提升时补符号位 （符号位：截断后的最高位）

所以刚刚的代码应该是这样计算的：



注意：

如果在提升之后要按照 %d 的格式打印时，那么把它看位一个有符号数的话，你需要判断它到底需不需要求**源码**