1. 1) int x=-32768, 2) short y=522, 3) unsigned z=65530
   4) char c='@', 5) float a=-1.1, 6) double b=10.5,
   7) float u = 123456.789e4, 8) double v= 123456.789e4

变量	x	y	z	c
机器数	0xffff8000	0x20a	0xfffa	0x40
变量	a	b	u	v
机器数	0xbf8ccccd	0x4025000000000000	0x4e932c06	0x41d26580b4800000

```
(gdb) x/1xw &x
0x7fffffffdd40: 0xffff8000
(gdb) x/1xh &y
0x7fffffffdd36: 0x020a
(gdb) x/1xh &z
0x7fffffffdd44: 0xfffa
(gdb) x/1xb &c
0x7fffffffdd35: 0x40
(gdb) x/1xw &a
0x7fffffffdd38: 0xbf8ccccd
(gdb) x/1xg &b
0x7fffffffdd48: 0x4025000000000000
(gdb) x/1xw &u
0x7fffffffdd3c: 0x4e932c06
(gdb) x/1xg &v
0x7fffffffdd50: 0x41d26580b4800000
```

```
Starting program: /home/river/Desktop/workspace/lab02/test
++++++++++Machine value++++++++++++++++
x = 0xffff8000
y = 0x20a
z = 0xfffa
c = 0x40
a = 0xbf8ccccd
b = 0x4025000000000000
u = 0x4e932c06
v = 0x41d26580b4800000
++++++++++Real value++++++++++++++++++
x = -32768
y = 522
z = 65530
c = @
a = -1.100000
b = 10.500000
u = 1234567936.000000
v = 1234567890.000000
[Inferior 1 (process 20775) exited normally]
```

运行代码验证输出与 gdb 查看的结果一致。

2. 1) 使用 gdb 命令查看程序变量的取值，填写下面两个表格：

a 的存放地址(&a)	b 的存放地址(&b)	x 的存放地址(&x)	y 的存放地址(&y)
0x7fffffffdd50	0x7fffffffdd54	0x7fffffffdd38	0x7fffffffdd30

执行步数	x 的值(机器)	y 的值(机器)	*x 的值(真值)	*y 的值(真值)
第一步前	0x7fffffffdd50	0x7fffffffdd54	1	2
第一步后	0x7fffffffdd50	0x7fffffffdd54	1	3
第二步后	0x7fffffffdd50	0x7fffffffdd54	2	3
第三步后	0x7fffffffdd50	0x7fffffffdd54	2	1

```
river@ubuntu:~/Desktop/workspace/lab02$ gdb -q swap
Reading symbols from swap...done.
(gdb) b 2
Breakpoint 1 at 0x676: file swap.c, line 2.
(gdb) b 3
Breakpoint 2 at 0x68a: file swap.c, line 3.
(gdb) b 4
Breakpoint 3 at 0x69e: file swap.c, line 4.
(gdb) b 5
Breakpoint 4 at 0x6b2: file swap.c, line 5.
(gdb) b 11
Breakpoint 5 at 0x6da: file swap.c, line 11.
(gdb) run
Starting program: /home/river/Desktop/workspace/lab02/swap

Breakpoint 5, main () at swap.c:11
11          xor_swap(&a, &b);
(gdb) x/1xw &a
0x7fffffffdd50: 0x00000001
(gdb) x/1xw &b
0x7fffffffdd54: 0x00000002
(gdb) c
Continuing.

Breakpoint 1, xor_swap (x=0x7fffffffdd50, y=0x7fffffffdd54) at swap.c:2
2           *y=*x ^ *y; /* 第一步 */
(gdb) x/1xw &x
0x7fffffffdd38: 0xffffdd50
(gdb) x/1xw &y
0x7fffffffdd30: 0xffffdd54
(gdb) x/1xw x
0x7fffffffdd50: 0x00000001
(gdb) x/1xw y
0x7fffffffdd54: 0x00000002
(gdb) p *x
$1 = 1
(gdb) p *y
$2 = 2
(gdb) c
Continuing.

Breakpoint 2, xor_swap (x=0x7fffffffdd50, y=0x7fffffffdd54) at swap.c:3
3           *x=*x ^ *y; /* 第二步 */
(gdb) x/1xw x
0x7fffffffdd50: 0x00000001
(gdb) x/1xw y
0x7fffffffdd54: 0x00000003
```

```
(gdb) p *x
$3 = 1
(gdb) p *y
$4 = 3
(gdb) c
Continuing.

Breakpoint 3, xor_swap (x=0x7fffffffdd50, y=0x7fffffffdd54) at swap.c:4
4           *y=*x ^ *y; /* 第三步 */
(gdb) x/1xw x
0x7fffffffdd50: 0x00000002
(gdb) x/1xw y
0x7fffffffdd54: 0x00000003
(gdb) p *x
$5 = 2
(gdb) p *y
$6 = 3
(gdb) c
Continuing.

Breakpoint 4, xor_swap (x=0x7fffffffdd50, y=0x7fffffffdd54) at swap.c:5
5       }
(gdb) x/1xw x
0x7fffffffdd50: 0x00000002
(gdb) x/1xw y
0x7fffffffdd54: 0x00000001
(gdb) p *x
$7 = 2
(gdb) p *y
$8 = 1
(gdb) c
Continuing.
[Inferior 1 (process 21509) exited normally]
```

2) 运行 reverse.c，并说明输出这种结果的原因，修改代码以得到正确的逆序数组

```c
#include <stdio.h>

void xor_swap(int *x, int *y) {
    *y=*x ^ *y; /* 第一步 */
    *x=*x ^ *y; /* 第二步 */
    *y=*x ^ *y; /* 第三步 */
}

void reverse_array(int a[], int len) {
    int left, right=len-1;
    for (left=0; left<right; left++, right--)
        xor_swap(&a[left], &a[right]);
}

int main()
{
    int a[] = {1,2,3,4,5,6,7};
    reverse_array(a, 7);
    int i;
    for(i = 0; i < 7; ++i)
        printf("%d ", a[i]);
    printf("\n");
}
```

```
river@ubuntu:~/Desktop/workspace/lab02$ vi reverse.c
river@ubuntu:~/Desktop/workspace/lab02$ gcc reverse.c -o reverse
river@ubuntu:~/Desktop/workspace/lab02$ ./reverse
7 6 5 0 3 2 1
river@ubuntu:~/Desktop/workspace/lab02$ vi reverse.c
river@ubuntu:~/Desktop/workspace/lab02$ gcc reverse.c -o reverse
river@ubuntu:~/Desktop/workspace/lab02$ ./reverse
7 6 5 4 3 2 1
```

循环条件是 left<=right，即 left==right==3 时执行 xor_swap(&a[3], &a[3])
两个相同的二进制数进行异或操作得到 0，则执行三次 a[3]=a[3] ^ a[3]后 a[3]的值为 0

3. 解释语句输出为 False 的原因并填写在表格中

```
river@ubuntu:~/Desktop/workspace/lab02$ vi tf.c
river@ubuntu:~/Desktop/workspace/lab02$ gcc tf.c -o tf
river@ubuntu:~/Desktop/workspace/lab02$ ./tf
++++++++++True or False++++++++++++++++
x==(int)xd True
x==(int)xf False
p1!=p2 False
result1==d True
result2==d False
```

```
(gdb) p x
$36 = 2147483647
(gdb) p xf
$37 = 2.14748365e+09
(gdb) p (int)xf
$38 = -2147483648
(gdb) p p1
$39 = 3.14159274
(gdb) p p2
$40 = 3.14159274
(gdb) p f
$41 = 1.00000002e+20
(gdb) p (double)f
$42 = 1.0000000200408773e+20
(gdb) p (d+f)
$43 = 1.0000000200408773e+20
```

	输出	原因
语句一	True	
语句二	False	int 有 32 位精度，而 float 只有 24 位，将 INT_MAX 赋值给 float xf 时有数据被舍入，得 2.1478365e+09 大于 2147483647，强制转化为 int 时符号位是 1，识别为负数，原 int x 为正数
语句三	False	3.141592653 和 3.141592654 所表示的精度均超出 float 所能表示的范围，p1 与 p2 只精确到 3.14159274，相等
语句四	True	
语句五	False	执行(d+f)-f 时 f 由 float 强制转化为 double，1.00000002e+20 变为 1.0000000200408773e+20，d 相对 f 数值过小，相加时可忽略不计，double f 与(d+f)近似相等，result2 为 0

4. 请在单步运行过程中完成下面的表格：

	机器数 (十六进制)	真值 (十进制)		机器数 (十六进制)	真值 (十进制)
x	0x66	102	y	0x39	57
~x	0xffffff99	4294967193	!x	0xffffff00	4294967040
x & y	0x20	32	x && y	0x1	1
x | y	0x7f	127	x || y	0x1	1

	机器数 (十六进制)	真值 (十进制)	OF	SF	CF	AF
x1	0x7fffffff	2147483647	0	0	0	0
y1	0x1	1	0	0	0	0
sum_x1_y1	0x80000000	2147483648	1	1	0	1
diff_x1_y1	0x7ffffffe	2147483646	0	0	0	0
diff_y1_x1	0x80000002	2147483650	0	1	1	1
x2	0x7fffffff	2147483647	0	0	0	0
y2	0x1	1	0	0	0	0
sum_x2_y2	0x80000000	2147483648	1	1	0	1
diff_x2_y2	0x7ffffffe	2147483646	0	0	0	0
diff_y2_x2	0x80000002	2147483650	0	1	1	1

2) 写出上面表格中每个标识位变化的原因，可直接在上表中注明。

sum_x1_y1: 01111111111111111111111111111111
+ 00000000000000000000000000000001
= 10000000000000000000000000000000,
结果溢出了，OF 为 1，sum 符号为 1，SF 为 1，第三位发生进位，AF 为 1。
diff_y1_x1: 00000000000000000000000000000001
- 01111111111111111111111111111111
= 10000000000000000000000000000010
sum 符号为 1，SF 为 1，最高有效位发生借位，CF 为 1，第三位发生借位，AF 为 1
sum_x2_y2: 01111111111111111111111111111111
+ 00000000000000000000000000000001
= 10000000000000000000000000000000,
结果溢出了，OF 为 1，sum 符号为 1，SF 为 1，第三位发生进位，AF 为 1。
diff_y2_x2: 00000000000000000000000000000001
- 01111111111111111111111111111111
= 10000000000000000000000000000010
sum 符号为 1，SF 为 1，最高有效位发生借位，CF 为 1，第三位发生借位，AF 为 1