```c
#include<stdio.h>

/* 2.62 */
int int_shifts_are_arithmetic()
{
    /*
     * 算术位移会在负数位移时在高位补 1，逻辑位移则为补 0
     * 若为前者，位移后仍为负值，否则位移后会变为正数
     * 故只需判断位移时后的正负性，即为函数返回值
     */
    int a = -1;
    return (a >> 1) < 0;
}

/* 2.67 */
int int_size_is_32()
{
    /*
     * A. C语言标准没有规定 (x << k) 操作在 k >= w (w表示x的数据类型的位数）时该如何表现，不同机器采用不同的做法，
     * 在某些机器上（例如本题的）可能会选择对 k 作 k mod w 得到真正的位移量
     * 此时若 k == 32, 则位移量为 0 ，beyond_msb 并未如期赋值为 0 ，发生错误
     *
     * B. 如下代码
     *
     * C. 如下代码
     *
     */

    /* 将 k >= w 的一次位移拆分成 k < w 的多次位移 */
    int set_msb = 1 << 15;
    set_msb = set_msb << 15;
    set_msb = set_msb << 1;
    int beyond_msb = set_msb << 1;
    /*
     * 在 w >= 32 的机器上 set_maxu 是非 0 数
     * 在 w <= 32 的机器上 beyond_maxu_ 是 0
     */
```

```c
    return set_msb && !beyond_msb;
}

/* 2.75 */
unsigned unsigned_high_prod(unsigned x, unsigned y)
{
    unsigned bits_of_int = sizeof(int) * 8;
    unsigned highest_bit_x = x >> (bits_of_int - 1);
    unsigned highest_bit_y = y >> (bits_of_int - 1);
    unsigned middle_term = highest_bit_x * y + x * highest_bit_y;
    /*
     * unsigned last_term = (highest_bit_x * highest_bit_y) << bits_of_int;
     * 左移 bits_of_int 本身会导致 last_term 超出限制溢出，预期情况下会使得低 bits_of_int 位全为 0，
     * 但由于 2.67 中描述的情况，位移量超过 bits_of_int 时的表现不确定
     * 且该项在本应是 signed_high_prod((int) x, (int)y) + middle_term + last_term 的结果中不影响低 bits_of_int 位
     * 即不影响结果
     * 故直接舍去这一项
     */
    return signed_high_prod((int)x, (int)y) + middle_term;
}

int main()
{
    printf("The result of int_shifts_are_arithmetic is %d\n", int_shifts_are_arithmetic());
    printf("The result of int_size_is_32 is %d\n", int_size_is_32());
    int a = 0, b = 0;
    printf("Input two unsigned integer to calculate unsigned_high_prod:\n");
    scanf("%u%u", &a, &b);
    printf("The result of unsigned_high_prod is %u\n", unsigned_high_prod(a, b));
    return 0;
}
```