Homework 2

Two's-Complement Encodings

Assume we have an integer type of 8 bits, fill in the table below.

Value	Two's complement
37	00100101
-15	11110001
85	01010101
-86	10101010

Shift Operation and Type Casting

Consider the following C expressions:

```
short s = -3;
unsigned short us = s;
int i = -52;
unsigned int ui = i;
```

Assume we are running code on **8-bit machine** using two's complement for singed integers. Also assume that right shift of signed values are performed **arithmetically**. A "short" integer is encoded 4 bits. Fill in the table below.

Expression	Binary Representation
us	1101
ui	11001100
us << 1	1010
i >> 2	11110011
ui >> 2	00110011
(short) i	1100
(int) s	11111101

Byte Order

Consider the following function

```
typedef unsigned char * byte_pointer;
void show_bytes(byte_pointer start, int len) {
    int i;
    for (i=0; i<len; i++)
    printf("%.2x", start[i]);
}
```

```
int val = 0x140A0233;
byte_pointer valp = (byte_pointer) & val;
```

What is the output of the following call to show_bytes on big-endian and little-endian machines respectively?

	little-endian	big-endian
`show_bytes(valp, 1);`	33	14
`show_bytes(valp, 2);`	3302	140a
`show_bytes(valp, 4);`	33020a14	140a0233

Unsigned Addition

Wirte a function with the following prototype:

```
/ * Determine whether arguments can be added without overflow
* This function should return 0 if arguments x and y can
* be added without causing overflow
*/
int uadd_ok(unsigned x, unsigned y);
```

```
int uadd_ok(unsigned x, unsigned y) {
    return x+y < x;
    /* or (return x+y < y;)
    */
}
```