Joshua Liu
105136031
CS 33
Discussion 1A

# Homework 1
Problems 2.72, 2.78, 2.82

## 2.72

A. The conditional test in the code always succeeds because sizeof(val) returns a value of type size_t, which is unsigned. Then, the expression will convert all values and results to unsigned values, causing the expression to always yield a value of 0 or greater. As a result, the conditional test in the code always succeeds.

B. Since the conditional statement is subtracting two numbers and relies on the value being negative for it to be false, we need to change the type returned by sizeof(val). We can do this by using (int) sizeof(val), as shown below completely.

```
void copy_int(int val, void *buf, int maxbytes)
{
	if (maxbytes - (int)sizeof(val) >= 0)
		memcpy(buf, (void *)&val, sizeof(val));
}
```

## 2.78

```
#include <limits.h>
int divide_power2(int x, int k)
{
	int biasValue = (1 << k) - 1;
	int sign_x = (x & INT_MIN) == INT_MIN;
	int bias = -sign_x & biasValue;

	return (x + bias) >> k;
}
```

Since we cannot use conditional statements, we use the bitwise operator & and other values to determine the sign of the integer x in order to know whether to include a bias. A bias is only applied to a negative number.
biasValue is the value of the bias that needs to be added to x if x is negative. sign_x determines whether x is positive or negative with the bitwise operator & and an equality check. Then, the bias that is added to x is determined, where it is 0 if x is positive, and the correct bias value if x is negative. We return the correct expression for dividing by the power of 2.

## 2.82

A. (x<y) == (-x>-y)
This expression yields 0 when x is INT_MIN, or Tmin. Tmin is equal to -Tmin, since the negative means to flip all the bits and add 1. As a result, the equality is no longer true in this scenario.

B. ((x+y)<<4) + y-x == 17*y+15*x
This expression always yields 1. We can simplify the left side and end up with the right:
((x+y)<<4) + y-x

x<<4-x+y<<4+y
x*16-x+y*16+y
15*x+17*y
17*y+15*x

C. ~x+~y+1 == ~(x+y)
This expression always yields 1. We can simplify the left side and end up with the right(we use the fact that -x = ~x + 1:
~x+~y+1
~x+1+~y+1-1
-x-y-1
-(x+y)-1
~(x+y)+1-1
~(x+y)

D. (ux-uy) == -(unsigned)(y-x)
This expression always yields 1.
The unsigned casting does not change the internal representation of the numbers. In other words, casting the numbers first then subtracting is the same as subtracting then casting the numbers to an unsigned integer. Then, we can see that the expression is the following:
(x-y) == -(y-x)
(x-y) == (x-y)

E. ((x >> 2) << 2) <= x
This expression always yields 1.
When we right shift x, we are getting rid of the two rightmost bits. Then, we are shifting it back to the left the same amount, resulting with two zero bits for the two rightmost bits. For a positive number, the two rightmost bits are going to be zero, meaning that its value must be smaller than it was before. For a negative number, since the two rightmost bits are zero, the value will be more negative than before. As a result, we can see that after this expression, it is always smaller or equal to x, whether it is positive or negative.