Valerie Hall

CSC 373

Homework 1

1. Is the result always 1 for result = (n1 < n2) == (-n1 > -n2)?

No the expression will not always evaluate to true. To determine this, I tested the extreme values and a couple unique values that could be substituted for n1 and n2. These values were INT\_MIN, INT\_MAX, 0, and -1. I tested all the combinations possible for substituting these values for both n1 and n2. The value that evaluated the result to 0 (false) was INT\_MIN. This is because when the processor tried to take the negative of a negative (which was INT\_MIN) it kept it as a negative instead of making it positive as would be expected. This made the left expression false and the right expression true which then evaluated to 0(false).

2. Is the result always 0 for result = ~n1 + ~n2 == ~(n1 + n2)?

Yes the result is always false. Once again I tested the extrema and other unique values in for n1 and n2. For each value the resulted evaluated to 0, or false.

3. Is the result always 1 for result = ((n1>>1) <<1) <= n1?

No the result is not always true. For example if you set n1=INT\_MIN. First you right shift n1, but by preserving the signed bit the right shift results in the same number of INT\_MIN because after the signed bit it is just zeros. Next doing a left shift to INT\_MIN (which is the same as doubling it) the result is 0. But 0 is not <= n1 because n1 is INT\_MIN (which is much less than 0) therefore the result is 0 (or false).

4. is the result always 1 for result = ((int) (u1 – u2))== - (n2 – n1)

Yes the result will always be 1, or true. After checking the extreme values INT\_MIN and INT\_MAX and the unique values 0 and -1 every combination of these values for n1, n2, u1, u2 resulted in a 1. Since these are the cases that would be thought to disrupt the expressions and they all resulted to true, then we can assume that it will always result in 1.

5. mystery1(unsigned int a, unsigned int b, unsigned char c)

a. mystery1 returns the value that’s in unsigned int a on the stack that the pointer is pointing to.

b. myestery1 puts unsigned int a (or 4 bytes) on the stack. Then assigns an unsigned char pointer to a which points to the first bit in a. Then b determines how much the pointer will be moved. If

b >= sizeof(unsigned int)) then b=0 and the pointer is not moved at all. Otherwise the pointer is moved for the value of b. Then the pointer is casted to an unsigned int and dereferenced which allows the function to return an unsigned int. The function then returns the value in the bit that the pointer was pointing to in a.

6. mystery2(unsigned int n)

a. mystery2 takes an unsigned int and switches its bits around and returns the value that the new order of bits equals. Some sample input/outputs that I got were: 16🡪268435456, -489🡪402587647, -1🡪-1, 117🡪1962934272, -3🡪-33554433. How it does this is first breaking down the 32 bits of the unsigned int temp into 4 bytes in the order B4 B3 B2 B1. Then after the shifting, &’s, and |’s the new order of the bytes would be B1B2B3B4.

b. (n >> 24) /\*This line shifts the bits of n right 24 spots. This means that the left 8 bits are now in the position where the first 8 bits used to be, and the left 24 bits are all zeros. In terms of the 4 bytes from letter a, B4 is now in the position of B1. \*/

((n<<8) & 0x00FF0000) /\* This line shifts the bits of n 8 spots to the left which makes the right eight bits zeros. In terms of the bytes, it shifts them all over a spot with the right replaced with zeros, or B3B2B100. Then the “&” makes all the bits zero except it preserves what is in B2 where the FF is in the hex address. \*/

((n>>8) & 0x0000FF00) /\* This line does almost the same thing as the previous line except it preserves the bits in the second byte. After the right shift the order of the bytes is 00B4B3B2. So this time by using the “&” with the hex address it is preserving the bits in B3. \*/

(n<<24) /\*This line does something similar to the first line. It shifts the bits left 24 spots which means that the right 8 bits are in the spot where the left 8 bits used to be and the right 24 bits are all zeros. So now B1 is in the position of B4.

In terms of the 4 eight bit bytes:

Before: B1 B2 B3 B4

After: B4 B3 B2 B1