1. **Finish your homework independently**
2. **Convert this docx to pdf: “stuID_name_csapp1.pdf”**
   Example: ” 2017010000_zhangsan_csapp1.pdf”
3. **Submit this pdf: learn.tsinghua.edu.cn**

2.64 ◆

Write code to implement the following function:

```
/* Return 1 when any even bit of x equals 1; 0 otherwise.
   Assume w=32. */
int any_even_one(unsigned x){
   // your code
}
```

Your function should follow the bit-level integer coding rules, except that you may assume that data type int has $w = 32$ bits.

```
/* Return 1 when any even bit of x equals 1; 0 otherwise.
   Assume w=32 */
int any_even_one(unsigned x){
    return !!(0x55555555&x);
}
```

2.73 ◆◆

Write code for a function with the following prototype:

```
/* Addition that saturates to TMin or TMax */
int saturating_add(int x, int y){
  // your code
}
```

Instead of overflowing the way normal two's-complement addition does, saturating addition returns *TMax* when there would be positive overflow, and *TMin* when there would be negative overflow. Saturating arithmetic is commonly used in programs that perform digital signal processing.

Your function should follow the bit-level integer coding rules.

```
/* Addition that saturates to TMin or TMax */
int saturating_add(int x,int y){
    int TMax = INT_MAX, TMin = INT_MIN;
    int s = x+y;
    int pos = (~x&TMin) && (~y&TMin) && (s&TMin); // positive overflow
    int neg = (x&TMin) && (y&TMin) && (~s&TMin); // negative overflow
    int s1 = (-pos)&TMax; // -pos==0xffffffff when pos==1
    int s2 = (-neg)&TMin; // -neg==0xffffffff when neg==1
    int s3 = (pos-1)&(neg-1)&s; // pos-1==0xffffffff when pos==0 (so as neg)
    return s1+s2+s3;
}
```

2.81 ◆

We are running programs on a machine where values of type int are 32 bits. They are represented in two' s complement, and they are right shifted arithmetically. Values of type unsigned are also 32 bits.

We generate arbitrary values x and y, and convert them to unsigned values as follows:

```
/* Create some arbitrary values */
int x = random();
int y = random();
/* Convert to unsigned */
unsigned ux = (unsigned) x;
unsigned uy = (unsigned) y;
```

For each of the following C expressions, you are to indicate whether or not the expression *always* yields 1. If it always yields 1, describe the underlying mathematical principles. Otherwise, give an example of arguments that make it yield 0.

A. (x>y) == (-x<-y)

不恒为 1。

反例： `x=TMin, y=0` 则 `(x>y)==0 && (-x<-y)==1`

B. ((x+y)<<5) + x-y == 31*y+33*x

恒为 1。

一方面因为算术左移 5 位在不溢出的情况下，相当于乘 $2^5 = 32$；另一方面，有符号整数的加减乘相当于在 $[-2^{w-1}, 2^{w-1})$ 范围内的模 2^w 意义下进行运算，因此在溢出的情况下依然恒等。

C. ~x+~y == ~(x+y)

不恒为 1。

反例：`x=0, y=-1`，则 `~x+~y==-1 && ~(x+y)==0`

D. (int)(ux-uy) == - (y-x)

恒为 1。

首先因为有符号整数的加减运算相当于在 $[-2^{w-1}, 2^{w-1})$ 范围内的模 2^w 意义下进行，因此 `-(y-x)==x-y`；其次，有符号和无符号整数在加法下有群同构：$\mathrm{TAdd}_w(u,v) = \mathrm{U2T}(\mathrm{UAdd}_w(\mathrm{T2U}(u),\mathrm{T2U}(v)))$，因此 `(int)(ux-uy)==x-y`

E. ((x >> 1) << 1) <= x

恒为 1。

因为算术右移总是向 $-\infty$ 取整。

(PAGE 81)

Bit-level integer coding rules

In several of the following problems, we will artificially restrict what programming constructs you can use to help you gain a better understanding of the bit-level, logic, and arithmetic operations of C. In answering these problems, your code must follow these rules:

. Assumptions

Integers are represented in two’ s-complement form.

Right shifts of signed data are performed arithmetically.

Data type int is w bits long. For some of the problems, you will be given a specific value for w, but otherwise your code should work as long as w is a multiple of 8. You can use the expression sizeof(int)<<3 to compute w.

. Forbidden

Conditionals (if or ?:), loops, switch statements, function calls, and macro invocations.

Division, modulus, and multiplication.

Relative comparison operators (<, >, <=, and >=).

Casting, either explicit or implicit.

. Allowed operations

All bit-level and logic operations.

Left and right shifts, but only with shift amounts between 0 and $w - 1$.

Addition and subtraction.

Equality (==) and inequality (!=) tests. (Some of the problems do not allow these.)

Integer constants INT_MIN and INT_MAX.