Note 1: In all assignments that asks for writing a program, it should contain at least one test scenario in addition to the main functionality of your program.

Note 2: 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:

Bit-level integer coding 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.

Even with these rules, you should try to make your code readable by choosing descriptive variable names and using comments to describe the logic behind your solutions. As an example, the following code extracts the most significant byte from integer argument x:

```
/* Get most significant byte from x */
int get_msb(int x) {
        /* Shift by w-8 */
        int shift_val = (sizeof(int)-1)<<3;
        /* Arithmetic shift */
        int xright = x >> shift_val;
        /* Zero all but LSB */
        return xright & 0xFF;
}
```

For answers to Q1, Q2, Q3, submit separate .c files.

Q1. (10 points)

Write code to implement the following function:

/* Return 1 when any odd bit of x equals 1; 0 otherwise.
Assume w=32. */
int any_odd_one(unsigned x);

Your function should follow the **bit-level integer coding rules**, except that you may assume that data type int has $w = 32$ bits.
Assume the least significant bit would have index 0, which would make it an even bit. Consequently, the bit after that at index 1 would be an odd bit.

Q2. (20 points)

An IPv4 address is composed of **four bytes** typically separated by dots in presentation, e.g. 182.24.45.34. IP addresses are categorized into several classes. Assume IP address classes are as shown below.

IP Address Classes

A	0.0.0.0 to 127.255.255.255	First bit == 0
B	128.0.0.1 to 191.255.255.255	First bits == 1 0
C	192.0.0.0 to 223.255.255.255	First bits == 1 1 0
D	224.0.0.0 to 239.255.255.255	First bits == 1 1 1 0
E	240.0.0.0 to 255.255.255.255	First bits == 1 1 1 1 0

Notice that there is a pattern on first bit(s) for IP addresses in each class if you examine the binary representation of the first number.

Typically routers makes decisions based on IP class.

Write a C function *char getIPClass(unsigned int)* that receives an IP address in form of an unsigned int as the input parameter and return the class of that IP address as a character. We are assuming each byte of our unsigned input contains one part of IP address. For example, IP address 182.24.45.34, which belong to class B, will be stored in variable i as follows.
unsigned i = 0xB6182D22; //
char IPClass (unsigned int);

Your function should follow the **bit-level integer coding rules, except that you can use if and comparison (e.g., <, <=, ==) in your return statements**. There is also a version that you can write without if statements.

Q3. (10 points)

Write **C expressions** to generate the bit patterns that follow, where a_k represents k repetitions of symbol a. Assume a w-bit data type. Your code may contain references to parameters `j` and `k`, representing the values of `j` and `k`, but **not** a parameter representing w. Using constants such as INT_MIN and INT_MAX is not allowed since it is an indirect reference to the value of w.

Bit Pattern A. $1_{w-k}0_k$
Bit Pattern B. $0_{w-k-j}1_k0_j$

Your C expressions should be programmed in .c file with sample tests.
Your expression should follow the **bit-level integer coding rules.** This means solutions including *loops* are **not** valid.

Help:
For bit pattern 1_w , the answer is printf("%x\n", -1);

For Q4, Q5 and Q6, submit a .pdf file.

Q4. (20 points) Consider a machine with a 5-bits word size. What are the values of these?

	Binary	Decimal
UMin	00000	0
UMax	11111	31
TMin	10000	-16
TMax	01111	15

Q5. **(20 points)** Consider a machine with a 4-bit word size. What is the result of $\texttt{UAdd}_4(u, v)$ $\texttt{TAdd}_4(u, v)$, $\texttt{UMult}_4(u, v)$, and $\texttt{TMult}_4(u, v)$ for $\texttt{u} = 1000_2$ and $\texttt{v} = 1101_2$? Represent the final result in both **binary** and **decimal**.

	binary	Decimal
$\text{UAdd}_4(u , v)$	0101	5
$\text{TAdd}_4(u , v)$	0101	5
$\text{UMult}_4(u , v)$	1000	8
$\text{TMult}_4(u , v)$	1000	-8

Q6. **(20 points)** Assume the following definitions for x and y, complete the results of following shifts for x and y in the table. Write your result in four digit hex.

short int x = 0xFFFA; unsigned short y = 0xFFFA;

x <<2	0xFFE8
x>>3	0xFFFF
y<<1	0xFFF4
y>>2	0x3FFE