1. You have the following *C* variable declarations in a *C* program.

```
int a = 1, b = 2, c = 3;
int e[] = {4, 5, 6};
int *f[] = {e, &a, &b, &c};
int **g = &f[1];
```

For each of the following `printf` statements, indicate what gets printed.

(a) `printf( "%d\n", *e );`

(b) `printf( "%d\n", **g );`

(c) `printf( "%d\n", **(f + 2) );`

(d) `printf( "%d\n", *(e + 2) );`

(e) `printf( "%d\n", *g[2] );`

(f) `printf( "%d\n", *(f[0] + 1) );`

2. For the *C* expressions on the left, find the description on the right that **best** matches it. You may assume that where the ellipses (...) occur, there is code that assigns an arbitrary value to `a`.

1. `int a;`
 `...`
 `a  = a | (0xf << 5);`
	A. Set bits 5 and 8 of `a`. Keep others the same.

2. `int a;`
 `...`
 `a &= ~((1 <<5) | (1`
 `<<8));`
	B. Clear bits 5 and 8 of `a`. Keep others the same.

3. `int a;`
 `...`
 `a |= (1 << 5) | (1 << 8);`
	C. Set bits 5 through 8 of `a`. Keep others the same.

4. `int a;`
 `...`
 `a = 0b1001 << 5;`
	D. Set bits 5 through 8 of `a` to 5. Keep others the same.

5. `int a;`
 `...`
 `a >>= 8;`
 `a &= 0x01;`
	E. Assign `a` such that it is all zeros except for bits 5 and 8.

6. `int a;`
 `...`
 `a ^= (1 << 8);`
	F. Move bit 8 of `a` to least-significant bit, all others zero.

7. `int a;`
 `...`
 `a = a & ~(0xf << 5);`
 `a |= (0b0101 << 5);`
	G. Invert bit 8 of a, keep others the same.

3. Suppose you are using a Raspberry Pi and you need a third UART (serial line) to transmit serial data only. You decide to use GPIO pin 23 as a mock serial output since you are not using it for anything else. You must use memory-mapped I/O direct to the RPi registers only, i.e., no device drivers or other libraries. Do not use C macros other than those provided.

(a) Complete the following function (in C) to initialize pin 23 appropriately. Use the macros and function declaration provided.

```
#define GPIO ((volatile unsigned int *)0x3f200000) // gpio base


void initMockSerial() {
      // your code inserted here

}
```

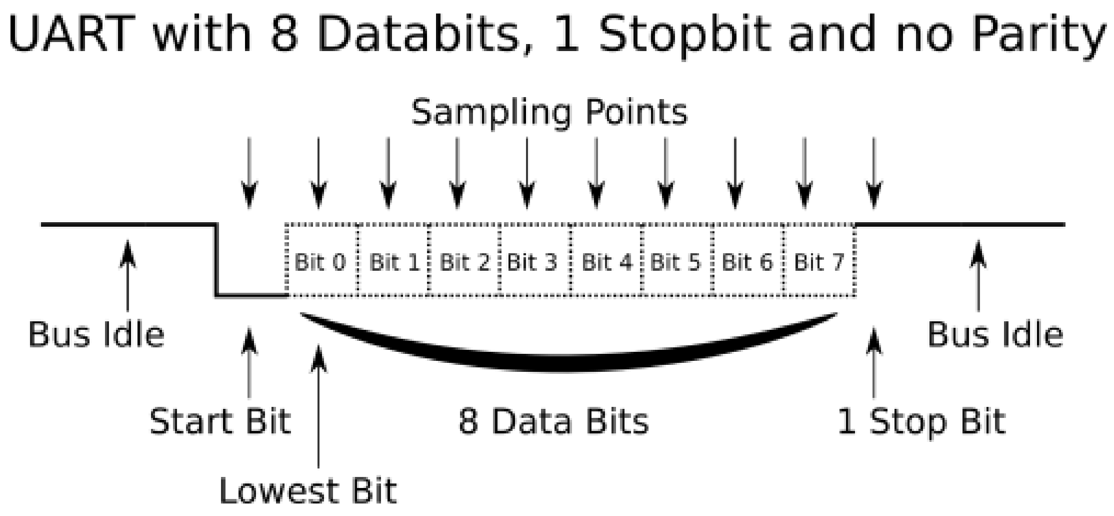
(b) Complete the following function (in C) to transmit a single character at 28800*baud*. Use the macros and function declaration provided. You may assume that the function *timedWait* already exists to wait for a period of time specified in microseconds. Assume 1 start bit, 1 stop bit, 8 bits data, and no parity. Note that 1/28800 = 34.7e-6 approximately.

```
void timedWait( int delay ); // delay in usec

void sendMockSerial( unsigned char c ) {
     // your code inserted here

}
```

The following figure may be helpful.



4. Suppose you are using a Raspberry Pi to control a robot, and that the robot has an “LV-MaxSonar-EZ1” sonar range sensor on board. Besides ground and power, there are two pins you will use to interact with the range sensor. The first pin, **RX**, turns range find on and off. Given that you plan to use multiple sensors, and that they can interfere with each other, you opt to turn off the sensor until your program is ready to acquire a range. That is, **RX** should normally be **LOW** until ready to start a range measurement. The second pin, **PW**, outputs the range as a pulse width. You will use GPIO pins 9 and 10 to communicate with the sonar pins **RX** and **PW** respectively.

   You must use memory-mapped I/O direct to the RPi registers only, i.e., no device drivers or other libraries. You may not use C macros other than those provided in the question.

   To acquire a range measurement, follow these steps.

1. set **RX** high
2. wait for **PW** to go high
3. time delay until **PW** goes low
4. reset **RX** to low

(a) Complete the following function (in C) to initialize pins 9 and 10 appropriately. Your function should also initialize the mapping of the system timer registers so that the system timer counter can be read from C as a 64-bit `long int`, reference by `*clock`.

```
#define GPIO ((volatile unsigned int *)0x3f200000) // gpio base
#define CLO ((volatile unsigned int *)(MMIO_BASE + 0x3f003004))
#define CHI ((volatile unsigned int *)(MMIO_BASE + 0x3f003008))

void initSonar() {
      // your code inserted here

}
unsigned long getSystemTimerCounter() {
    unsigned int h=-1, l;
    h=*CHI; l=*CLO;
    if(h!=*CHI) { h=*CHI; l=*CLO; }
    return ((unsigned long) h << 32) | l;
}
```

(b) Complete the following function (in C) to take a range measurement. Your function should return a range in inches (truncated to the nearest integer), using the scale 147*uS* per *inch*. Note that you might not need to use the *timedWait* function provided, but it may prove to be a useful reference.

```
int getSonar() {
      // your code inserted here
}
```

5. Repeat question 4 using interrupts for PW. Include the interrupt handler and initialization. You may assume that you have the following functions available.

```
void initSonar(); // your solution for 4(a)
void enRisingInt( int pin ); // enables interrupt for pin on rising edge
void enFallingInt( int pin ); // enable interrupt for pin on falling edge
void enInterrupts(); // enable CPU interrupts
void disInterrupts();  // disable CPU interrupts
int isInt( int pin );  // returns non-zero if interrupt occurred for pin
void clearInt( int pin ); // clears interrupt for pin
```