August 31, 2020

1. a) 0

Notes

- a) is 0 because (i >> 1 + j >> 1 = i >> 10 >> 1 = 0)
- Bitwise Shift Operators
 - has lower precedence than arithematic operators

Example:

```
i << 2 + 1 means i << (2+1) and not (i << 2) + 1
```

- << : Left Shift
- >> : Right Shift
- Tip: Always shift only on unsigned numbers for portability

Example

- >> = / << = : Are bitwise shift equivalent of + = (

b) 0

Notes

- i is 111111111111111
- i is 000000000000000
- so i & i = 0
- : Bitwise complement (NOT)

a	~ a
0	1
1	0

Example:

```
1 0 1 1 1 //<- this is 7
2 -------
3 1 0 0 0 //<- this is 8
4
5 so, ~ 7 = 8
```

• &: Bitwise and

a	b	a & b
0	0	0
0	1	1
1	0	0
1	1	1

Example:

```
0 1 1 1 //<- this is 7
0 1 0 0 //<- this is 4
3 ------
4 0 1 0 0 //<- this is 4
5
6 so, 7 & 4 = 4
```

- ullet : Bitwise exclusive or
- ullet |: Bitwise inclusive or
- c) 1

Notes

- i is 111111111111110
- j is 000000000000000
- $\bullet\,$ i & j is 0000000000000000 or 1
- i & j ^ k is 1

• ^: Bitwise XOR

a	b	a ^ b
0	0	0
0	1	1
1	0	1
1	1	0

Example:

```
1 0 1 1 1 //<- this is 7
2 0 1 0 0 //<- this is 4
3 ------
4 0 0 1 1 //<- this is 3
5
6 so, 7 ^ 4 = 3
```

d) 0

Example

- i is 000000000000111
- j is 000000000001000
- \bullet i ^ j is 0000000000000000 or 0
- k is 000000000001001
- i ^ j & k is 0000000000000000 or 0

Correct Solution

15

\underline{Notes}

• There is a precendence to the order of operations



e) • toggling from 0 to 1

```
i = 0x0000;
i |= 0x0001;
or
i |= 1 << 0; where i = 0x0000;
• toggling from 1 to 0

i = 0x0001;
i &= ~0x0001;
or
i &= ~(1 << 0); where i = 0x0001;</pre>
```

Correct Solution

• toggling from 0 to 1 of 4th bit

```
i = 0x0010;
i ^= 0x0000;
or
i ^= 1 << 4; where i = 0x0000;
• toggling from 1 to 0 of 4th bit

i = 0x0010;
i ^= 0x0010;
or
i ^= (1 << 4); where i = 0x0010;</pre>
```

Notes

- Toggling can be done using bitwise XOR
- Setting a bit
 - Is done using | or bitwise OR

- The idiom of above is $i = 1 \ll j$

- Clearing a bit
 - Is done using | or bitwise AND

- The idiom of above is i &= \sim (i << j)
- 2. It swaps the elements between x and y.

Notes

• Preprocessor performs operations of statements in order from left to right

#define
$$M(x,y)$$
 ((x)^=(y), (y)^=(x), (x)^=(y))

New value of y, using x from 1

New value of x, using y from 2, x from 1

3. #define MK_COLOR(r,g,b) (long) ((b | (g << 8)) | (b | (r << 16)))

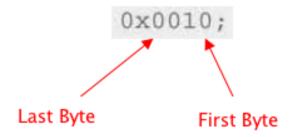
Rough Work

- 1. store b in bit 0
 - b
- 2. store g in bit 8
 - b | g << 8
- 3. store r in bit 16

```
\frac{Correct\ Solution}{\text{#define}\ MK\_COLOR(r,g,b)\ (long)\ ((\ref{eq:color} (g << 8)) \ |\ (\ref{eq:color} (b << 16)))}
```

Notes

 \bullet First Byte is furthest from 0x and first byte is closest to 0x



 $4. \qquad \bullet \ \ \mathsf{GET_RED}$

#define GET_RED(c) (long) (c & 0x007)

- GET_GREEN
- GET_BLUE