CSC 209 Review 9 Solution

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 <u>unsigned</u> numbers for portability

       ### Example

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

   b) 0

   ### Notes

- i is 1111111111111111
- i is 0000000000000000
- so i & i = 0
- : Bitwise complement (NOT)

a	$\sim$ 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:

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

- : Bitwise *exclusive or*
- |: Bitwise *inclusive or*

c) 1

Notes

- i is 1111111111111110
- j is 0000000000000000
- i & j is 0000000000000000 or 1
- i & j ^ k is 1

- ^: Bitwise XOR

a	b	a ˆ b
0	0	0
0	1	1
1	0	1
1	1	0

Example:

```
    0    1    1    1    //<- this is 7
    0    1    0    0    //<- this is 4
    --------------
    0    0    1    1    //<- this is 3

    so, 7 ^ 4 = 3

```

d) 0

Example

- `i` is 0000000000000111
- `j` is 0000000000001000
- `i ^ j` is 0000000000000000 or 0
- `k` is 0000000000001001
- `i ^ j & k` is 0000000000000000 or 0

Correct Solution

15

Notes

- There is a precendence to the order of operations

Highest: ~
             &
             ^

Lowest: |

e) • toglling from 0 to 1

```
i = 0x0000;
i |= 0x0001;
```

or

```
i |= 1 << 0;
```

Notes

- **Setting a bit**
  - Is done using | or bitwise OR

```
i = 0x0000;          /* i is now 0000000000000000 */
i |= 0x0010;          /* i is now 0000000000010000 */
```

  - The idiom of above is `i |= 1 << j`

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

```
i = 0x00ff;          /* i is now 0000000011111111 */
i &= ~0x0010;         /* i is now 0000000011101111 */
```

  - The idiom of above is `i &= ~(i << j)`