CSC 209 Review 9 Solution

August 31, 2020

1. a) 0

   <u>**Notes**</u>

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

       <u>**Example:**</u>

       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

       <u>**Example**</u>

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

   b) 0

   <u>**Notes**</u>

- ̃i is 1111111111111111
- i is 0000000000000000
- 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:

```
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

<u>Notes</u>

- ̃i is 1111111111111110
- j is 0000000000000000
- ̃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
7
```

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

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;
```

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;
```

Notes

• Toggling can be done using bitwise XOR
• **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)`

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 → New value → New value
   of x   of y   of x

3. **Notes**

   •