COMP 2401 -- Tutorial #3

Bit by bit

Learning Objectives

After this tutorial, you will be able to:

- use bitwise operations to get and store information in a single bit position
- retrieve and manipulate data from multi-dimensional arrays.

Tutorial

1. Download the file `T03.tar` from the tutorial page in *cuLearn*.

2. Implement the functions `getBit()`, `setBit()`, and `clearBit()` as prototyped.

3. Using the functions you just wrote, implement the function `printBits()` as prototyped. The output of `printBits('A')` should be:

```
0100 0001
```

4. Write code to accomplish the following:
   a. Clear the bit position 6 from all values in `arr`.
   b. Set the bit position 3 for all values in `arr`.
   c. Output all values in `arr`.

Exercises

1. Write a function `printIntBits(int c)`, which prints the bits of an integer parameter.

2. Write a function `printIntHex(int c)`, which prints the hexadecimal representation of an integer parameter. Don't forget to precede the number with '`0x`'.

3. Suppose we have an array `arr[m][n]`, which we would like to represent as a one-dimensional array `arrFlat[m*n]` (for example, `arr[5][5]` would be represented as `arrFlat[25]`)

   a. Write a function `index(int j, int k, int n)`, which returns a unique and valid index in the one-dimensional array for each valid pair `(j, k)` of indices in the `m` x `n` 2D array.

   b. Write the inverse functions `index_m(int i, int n)` and `index_n(int i, int n)` which give a unique, valid pair of indices in the `m` x `n` 2D array for each valid index in the 1D array.

   NOTE: if implemented correctly `index(index_m(a, n), index_n(a, n), n) == a`, as long as `a` is a valid index.

   c. Write similar functions for a 3D to 1D transformation.