```c
// bits.c ... functions on bit-strings

#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <assert.h>
#include "bits.h"
#include "util.h"

#define BITS_PER_WORD 32

typedef unsigned int Count;  // non-negative integer
typedef unsigned int Word;   // unsigned 32-bit integer value

typedef struct _BitsRep {
        Count  nbits;             // how many bits
        Count  nwords;            // how many Words in array
        Word   bitstring[1];  // array of Words to hold bits
                              // actual array size is nwords
} BitsRep;

// create a new Bits object

Bits newBits(int nbits)
{
        Count nwords = iceil(nbits,BITS_PER_WORD);
        Bits new = malloc(2*sizeof(Count) + nwords*sizeof(Word));
        assert(new != NULL);
        new->nbits = nbits;
        new->nwords = nwords;
        memset(&(new->bitstring[0]), 0x00, nwords*sizeof(Word));
        return new;
}

// release memory associated with a Bits object

void freeBits(Bits b)
{
        free(b);
}


// XXX put any helper functions here XXX
int ifloor(int val, int base)
{
        int floor = val / base;
        return floor;
}

// set the bit at position to 1
```

```c
void setBit(Bits b, int position)
{
        assert(b != NULL);
        assert(0 <= position && position < b->nbits);

        // XXX complete this function XXX
    int wordIndex = ifloor(position, BITS_PER_WORD);
    int posInWord = position % BITS_PER_WORD;

    Word wordToSet = 0x1 << posInWord;

    b->bitstring[wordIndex] |= wordToSet;
}

// set all bits to 1
// including possibly some bits beyond the top-end of the bit-string

void setAllBits(Bits b)
{
        assert(b != NULL);
        memset(&(b->bitstring[0]), 0xFF, b->nwords*sizeof(Word));
}

// set the bit at position to 0

void unsetBit(Bits b, int position)
{
        assert(b != NULL);
        assert(0 <= position && position < b->nbits);

        // XXX complete this function XXX
    int wordIndex = ifloor(position, BITS_PER_WORD);
    int posInWord = position % BITS_PER_WORD;

    Word wordToSet = 0x1 << posInWord;

    b->bitstring[wordIndex] &= ~(wordToSet);

}

// set all bits to 0

void unsetAllBits(Bits b)
{
        assert(b != NULL);
        memset(&(b->bitstring[0]), 0x00, b->nwords*sizeof(Word));
}

// show Bits on stdout
// display in order MSB to LSB
// do not append '\n'
```

```
void showBits(Bits b)
{
        assert(b != NULL);

        // XXX complete this function XXX
    int bitsToNotShow = b->nwords*BITS_PER_WORD - b->nbits;
    for (int i = b->nwords-1; i >= 0; i--) {
        for (int j = BITS_PER_WORD - 1; j >= 0; j--) {
            Word mask = (1 << j);
            if (bitsToNotShow > 0) {
                bitsToNotShow--;
                continue;
            }
            if (b->bitstring[i] & mask)
                putchar('1');
            else
                putchar('0');
            bitsToNotShow--;
        }
    }
}
```