```
// bits.c ... functions on bit-strings
//
// A Bits value contains
// - length = how many bits are actually used
// - bits   = the actual 32-bit unsigned value

#include <assert.h>
#include <string.h>
#include "bits.h"

// check if the bit at position is 1

int bitIsSet(Bits b, int position)
{
        assert(0 <= position && position < b.length);
        uint mask = (1 << position);
        return ((b.bits & mask) != 0);
}

// set the bit at position to 1

Bits setBit(Bits b, int position)
{
        assert(0 <= position && position <= b.length);
        uint mask = (1 << position);
        b.bits = b.bits | mask;
        return b;
}

// set the bit at position to 0

Bits unsetBit(Bits b, int position)
{
        assert(0 <= position && position < b.length);
        uint mask = (~(1 << position));
        b.bits = b.bits & mask;
        return b;
}

// make a Bits value with nbits of 0 bits

Bits zeroBits(int nbits)
{
        Bits b;
        b.length = nbits;
        b.bits = 0;
        return b;
}

// how many usable bits in a Bits value
```

```
int nBits(Bits b)
{
        return b.length;
}

// return just the uint part of a Bits value
int bitsToInt(Bits b)
{
        return (int)b.bits;
}

// check whether a string is 0's and 1's only

bool isBits(char *str)
{
        while (*str != '\0') {
                if (*str != '0' && *str != '1')
                        return 0;
                str++;
        }
        return 1;
}

// convert a string of 0's and 1's to a Bits value

Bits strToBits(char *str)
{
        int i;
        Bits b;

        b.length = strlen(str);
        b.bits = 0x00000000;
        i = b.length - 1;
        while (*str != '\0') {
                if (*str == '1') {
                        b.bits |= (1 << i);
                }
                i--;
                str++;
        }
        return b;
}

// convert n-bit unsigned quantity to string
// place in a user-supplied buffer of length > n

void showBits(Bits b, char *buf)
{
        int i; char ch;
        uint bit = 0x00000001;

        buf[b.length] = '\0';
```

```
        for (i = b.length-1; i >= 0; i--) {
                ch = ((b.bits & bit) != 0) ? '1' : '0';
                buf[i] = ch;
                bit = bit << 1;
        }
}
```