

```
// buckets.c ... which buckets to examine
//
// Usage: ./buckets QueryHash
//
// The QueryHash value contains 0's, 1's and *'s and
// represents the multi-attribute hash value produced
// from a query like a,?,c
//
// Using bit-strings derived from this query hash, the
// program should print a list of buckets that will be
// examined in order to answer the query
//
// Bit-strings are written so that the most significant bit is
// on the left and the least significant bit is on the right.
//
// Example #1: consider a query "a,?,c" on a 32-page file
// where the choice vector is (0,0),(1,0),(2,0),(1,1),(0,1)
// assuming the query hash = 1*0*1 (depends on hash function)
// and the program would indicate buckets 17, 19, 25, 27
//
// Example #2: consider a query "x,y,?" on a 32-page file
// where the choice vector is (0,0),(1,0),(2,0),(1,1),(0,1)
// assuming the query hash = 01*01 (depends on hash function)
// and the program would indicate buckets 9, 13
//
// We do not consider hash functions, choice vectors or overflow
// pages in this question. This code is invoked after the bit
// strings are produced using the query, the hash function, and
// the choice vector

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

int main(int argc, char **argv)
{
    int i, j, nbits;
    Bits known, unknown;
    char out[40]; // output buffer for displaying bit-strings

    if (argc < 2) {
        fprintf(stderr, "Usage: ./buckets QueryHash\n");
        exit(EXIT_FAILURE);
    }

    nbits = strlen(argv[1]);
    assert(nbits > 0 && nbits < 32);

    // set up known and unknown bit-strings
```

```

known = zeroBits(nbits);  unknown = zeroBits(nbits);

j = nbits-1;
for (i = 0; i < nbits; i++,j--) {
    char c = argv[1][i];
    if (c == '1')
        known = setBit(known,j);
    else if (c == '0') {
        /* nothing to do */
    }
    else if (c == '*') {
        unknown = setBit(unknown,j);
    }
    else {
        fprintf(stderr, "Invalid QueryHash\n");
        exit(EXIT_FAILURE);
    }
}

showBits(known,out); printf("Known:   %s\n", out);
showBits(unknown,out); printf("Unknown: %s\n", out);

// calculate buckets to be examined

// TODO: add your code here

// how many *'s
int nstars = 0;
for (i = 0; i < nBits(unknown); i++) {
    if (bitIsSet(unknown,i)) nstars++;
}

// for all possible combinations of 2^nstars bits
int counter;
for (counter = 0; counter < (1<<nstars); counter++) {
    int i = 0, j = 0;
    Bits b = known;
    //showBits(b,out); printf("Starting with: %s\n",out);
    for (i = 0; i < nBits(unknown); i++) {
        //printf("checking for * at %d\n",i);
        if (bitIsSet(unknown,i)) {
            //printf("found * at %d\n",i);
            // fit next bit from counter into

hash
                                if (counter & (1<<j)) {
                                    //printf("counter has 1 at
%d\n",j);

                                    b = setBit(b,i);
                                }
                                j++;
        }
    }
}

```

```
    }  
    //showBits(b,out); printf("Bucket: %s\n",out);  
    printf("%d\n", bitsToInt(b));  
}  
  
return EXIT_SUCCESS;  
}
```