Question 1 (21 marks)

For this question, you will write code to implement a core component of a partial-match retrieval (PMR) multi-attribute hashing (MAH) query evaluator: generating a list of buckets to examine, based on a query hash.

Reminder:

- a PMR query is a conjunction of equality tests on a single table; values are supplied for some attributes, while other attributes are not specified (unknown)
- MAH uses bits from hash values for several attributes to determine an overall hash for a tuple or query; which bits of which attributes are used in the overall hash is determined by a choice vector

MAH Example: Consider a table `R(a,b,c)` with choice vector <(0,0),(1,0),(2,0),(0,1),(1,1),..>. This tells us that bit 0 (least significant) of the overall hash comes from bit 0 of the first attribute ("a"), bit 1 of the overall hash comes from bit 0 of the second attribute ("b"), and so on. If we insert a tuple like `(5,3,6)` into the table, and if `hash(5)=00101`, `hash(3)=00011` and `hash(6)=00110`, then the overall hash (assuming 5-bit hash values) will be `10011`, and so the tuple would be inserted into bucket 19.

PMR Query Examples: If we asked a query like `5,3,?`, then we would obtain a query hash `10*11`, where the `*` represents bits from the unknown attributes. To answer this query, we need to generate all possible values for the `*`'s, which would give `10011` and `10111`, and this gives us a list of buckets to check for matching tuples (in this case, buckets 19 and 23). If we asked a query like `5,?,6`, then we would obtain a query hash `*00*1` and would need to examine buckets `00001`, `00011`, `10001` and `10011` (i.e. buckets 1, 3, 17, 19).

You are to complete the implementation of a program called `buckets`, which takes as input (on the command line) a PMR query hash value, and writes one per line on the standard output, a list of buckets to examine. The `main()` function for `buckets` takes a query hash like `01*1*101*0` as its command-line argument, checks it, and converts it into two bit strings The first bit string looks contains the known bits of the MAH value, and looks like the query has with all `*`'s replace by 0. The second bit string contains 1 bits for all the `*`'s in the query hash, and 0 for all other bits (in other words, it tells you the positions of the `*`'s). As supplied, the `buckets` program simply prints these two bit strings and halts. Once you have completed it, the `buckets` program should also print a list of bucket numbers (in decimal) on its output.

There are examples of inputs to and expected outputs from `buckets` in the directory `q1/tests`, in the files with names like `t3` (query hash) and `t3.expected` (output).

Note: we are not considering linear-hashing here. Assume that all files will be of size 2^d, where d is the number of bits used in the hash values. The value of d is simply the length of the command-line argument to the `buckets` program.

The following files are available for the implementation and can be found in the `q1` directory in your exam environment:

- `Makefile` ... to control the compilation of the scanner
- `bits.h` ... interface to bit-string (Bits) operations

- `bits.c` ... implementation of bit-string (Bits) operations
- `buckets.c` ... main program (you modify this)

All of these files are complete and fully-functional except for `buckets.c`, which you are required to complete. You should not modify any of the other files in developing your solution.

If you run the `make` command, it will produce an executable file called `buckets`, which you could test using commands like tuples from the test files using e.g.

```
$ ./buckets 00**
Known:   0000
Unknown: 0011
0
1
2
3
$ ./buckets 1010*
Known:   10100
Unknown: 00001
20
21
```


At this stage, the only output you'll get is a display of the two bit-strings that `buckets` sets up before generating the bucket numbers.

Note that the data files are contained in a sub-directory called `tests`, along with the expected output for when a correct `buckets` program is run on them. Some hopefully useful notes about the data files:

- `t0` is a completely empty data file (0 bytes)
- `t1` contains a single page, but the page contains zero tuples
- `t2` contains a single page, with 15 tuples
- other data files contain a mixture of empty and non-empty pages
- offsets are relative to the start of the tuples, *not* the start of the Page
- each page is zero-filled when it is initially created (before tuples are added)

To help you check whether your program is working correctly, there is a script called `run_tests.sh` which will run your program against all of the tests and report the results, and add some output files to the `tests` directory which might help you to debug your code. Since `run_tests.sh` produces a lot of output (until your program is working), it might be useful to run it like:

```
$ sh run_tests.sh | less
```

Your task is to complete the `buckets.c` program. It requires around 20-30 lines of code to solve this problem; partial marks are available if you complete some of the code.

Some hints on how to approach this problem:

- take a quick look at the `bits.h` and `bits.c` file to see what it does
- then look at `buckets.c` to see what the current code does
- then plan how to generate the bucket numbers based on the known/unknown bits
- use `gdb`, if you know how; otherwise add plenty of `printf`'s for debugging

The directory `tests` contains data files (called `ti`) and an expected output file (called

`ti.expected`) from running `./buckets` on each data file. You should look at the expected output files to see what a correct program should produce.

Once your program can run under the `run_tests.sh` script with no errors, you can submit it. Note that we run your submission on additional tests for marking; table look-up answers are worth zero marks.

Instructions:

- Type your answer to this question into the file called `buckets.c`
- Submit via: **`give cs9315 sample_q1 buckets.c`**
  or via: Webcms3 > exams > Sample Exam > Submit Q1 > Make Submission

Hint: you may want to leave this question until you have completed all of the other questions.

End of Question