COMP9315 14s2

The University of New South Wales

COMP9315 DBMS Implementation Final Exam 14s2

DBMS Implementation

Question 1 (20 marks)

For this question, you will write functions to implement two of the major operations of an *iterator*. This iterator, called `Scan`, fetches all of the tuples from a file, getting the tuples in the order that they are stored in the file.

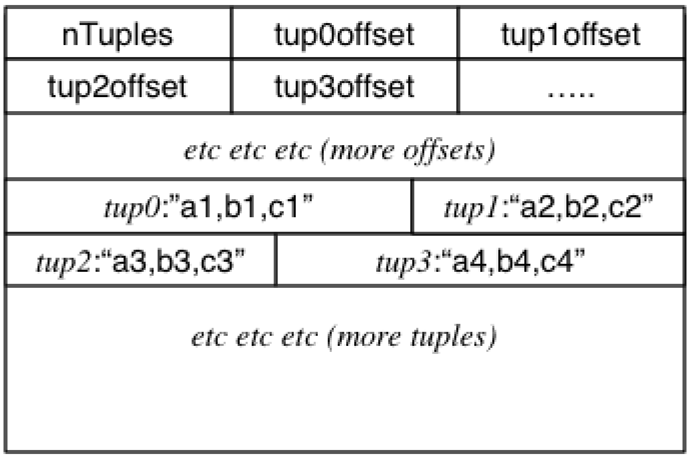
The data files used by this iterator have the following properties:

- a **file** is a sequence of *N* pages numbered 0..*N*-1
- a **page** contains a header and a sequence of *M* tuples, numbered 0..*M*-1
- a **tuple** is a sequence of characters, terminated by `'\0'` (i.e. a C string)

A page is a 1024-byte block, structured as follows (see `Page.h` for details)

- a header consisting of a 4-byte counter followed by an array of 63 offsets
- an array of bytes, containing tuple strings

The following diagram shows the detailed structure of each page:



The iterator has a main program which works as a tuple scanner. The `main()` function for the scanner takes a file name as a command-line argument, opens the file, starts the iterator and then fetches and prints all of the tuples in the file. The scanner also accesses the `Scan` data structure to print out the current page and current tuple number in the page of the fetched tuple. There are examples of the output from `scanner` in the directory `q1/tests`, in the files with names like `t3.expected`.

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
- `main.c` ... the main program of the scanner
- `Page.h` ... data structures for pages
- `Page.c` ... operations on pages
- `Scan.h` ... data structures for the iterator
- `Scan.c` ... operations on the iterator (incomplete)

All of these files are complete and fully-functional except for `Scan.c`, which you are required to complete.

If you run the `make` command, it will produce an executable file called `scanner`, which you could use to try to read the tuples from the test files using e.g.

```
$ ./scanner tests/t3
```

At this stage, the only output you'll get is "`Cannot start scan`". Once you have correctly implemented the iterator, you'll see the tuples instead.

Note that the data files are contained in a sub-directory called `tests`, along with the expected output for when a correct `scanner` 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 `check` 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 `check` produces a lot of output (until your program is working), it might be useful to run it like:

```
$ check | less
```

Your task is to complete the two functions `startScan()` and `nextTuple()` in `Scan.c`. You must preserve the function interfaces and you must use the supplied `Scan` structure. You are not allowed to modify any of the other files (except to add debugging code), and you cannot submit any file apart from `Scan.c`. It requires around 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 `main.c` file to see what it does and how it uses the `startScan()` and `nextTuple()` functions
- read the data structure definitions in `Scan.h` and `Page.h`
- start by implementing `startScan()`
- think carefully about the way `nextTuple()` moves from tuple to tuple and page to page
- use `gdb`, if you know how; otherwise add plenty of `printf`'s for debugging

The directory `tests` contains data files (called `t`*`i`*) and an expected output file (called `t`*`i`*`.out`) from running `./scanner` on each data file. You should look at the expected output files to see what a correct program should produce. The data files are in binary format, so can't be usefully viewed with `cat` or `less` or a text editor. If you want to look at them, try the `od` command (e.g. `od -c tests/t3`).

Once your program passes the `check` tests with no errors,you can submit it.

Instructions:

- Type your answer to this question into the file called `Scan.c`
- Submit via: **`submit q1`**
  (The `submit` command knows to collect the `Scan.c` file)

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

End of Question