THE UNIVERSITY OF SYDNEY

COMP2129 Assignment 2

Due: 6:00pm Friday, 1 May 2015

This assignment is worth 12% of your final assessment

Task description

In this assignment we will develop a key value store called **Snapshot DB** in the C programming language using dynamic data structures, ensuring that no memory errors occur. Each entry of the database is identified by a unique key string and contains a dynamically sized list of integer values.

You are encouraged to ask questions on ed using the assignments category. As with any assignment, make sure that your work is your own, and you do not share your code or solutions with other students.

Working on your assignment

You can work on this assignment on your own computer or the lab machines. Students can also take advantage of the online code editor on ed. Simply navigate to the assignment page and you are able to run, edit and submit code from within your browser. We recommend that you use Safari or Chrome.

It is important that you continually back up your assignment files onto your own machine, flash drives, external hard drives and cloud storage providers. You are encouraged to submit your assignment while you are in the process of completing it. By submitting you will obtain some feedback of your progress.

Academic declaration

By submitting this assignment you declare the following:

I declare that I have read and understood the University of Sydney Academic Dishonesty and Plagiarism in Coursework Policy, and except where specifically acknowledged, the work contained in this assignment or project is my own work, and has not been copied from other sources or been previously submitted for award or assessment.

I understand that failure to comply with the the Academic Dishonesty and Plagiarism in Coursework Policy, can lead to severe penalties as outlined under Chapter 8 of the University of Sydney By-Law 1999 (as amended). These penalties may be imposed in cases where any significant portion of my submitted work has been copied without proper acknowledgement from other sources, including published works, the internet, existing programs, the work of other students, or work previously submitted for other awards or assessments.

I realise that I may be asked to identify those portions of the work contributed by me and required to demonstrate my knowledge of the relevant material by answering oral questions or by undertaking supplementary work, either written or in the laboratory, in order to arrive at the final assessment mark.

I acknowledge that the School of Information Technologies, in assessing this assignment, may reproduce it entirely, may provide a copy to another member of faculty, and or communicate a copy of this assignment to a plagiarism checking service or in house computer program, and that a copy of the assignment may be maintained by the service or the School of IT for the purpose of future plagiarism checking.

Implementation details

Write a program in C that implements Snapshot DB as shown in the examples below. You can assume that our test cases will contain only valid input commands and not cause any integer overflows. Keys are case sensitive and do not contain spaces. Commands are case insensitive. Entry values are indexed from 1. Snapshots are indexed from 1 and are unique for the lifetime of the program. Keys, entries and snapshots are to be outputted in the order from most recently added to least recently added.

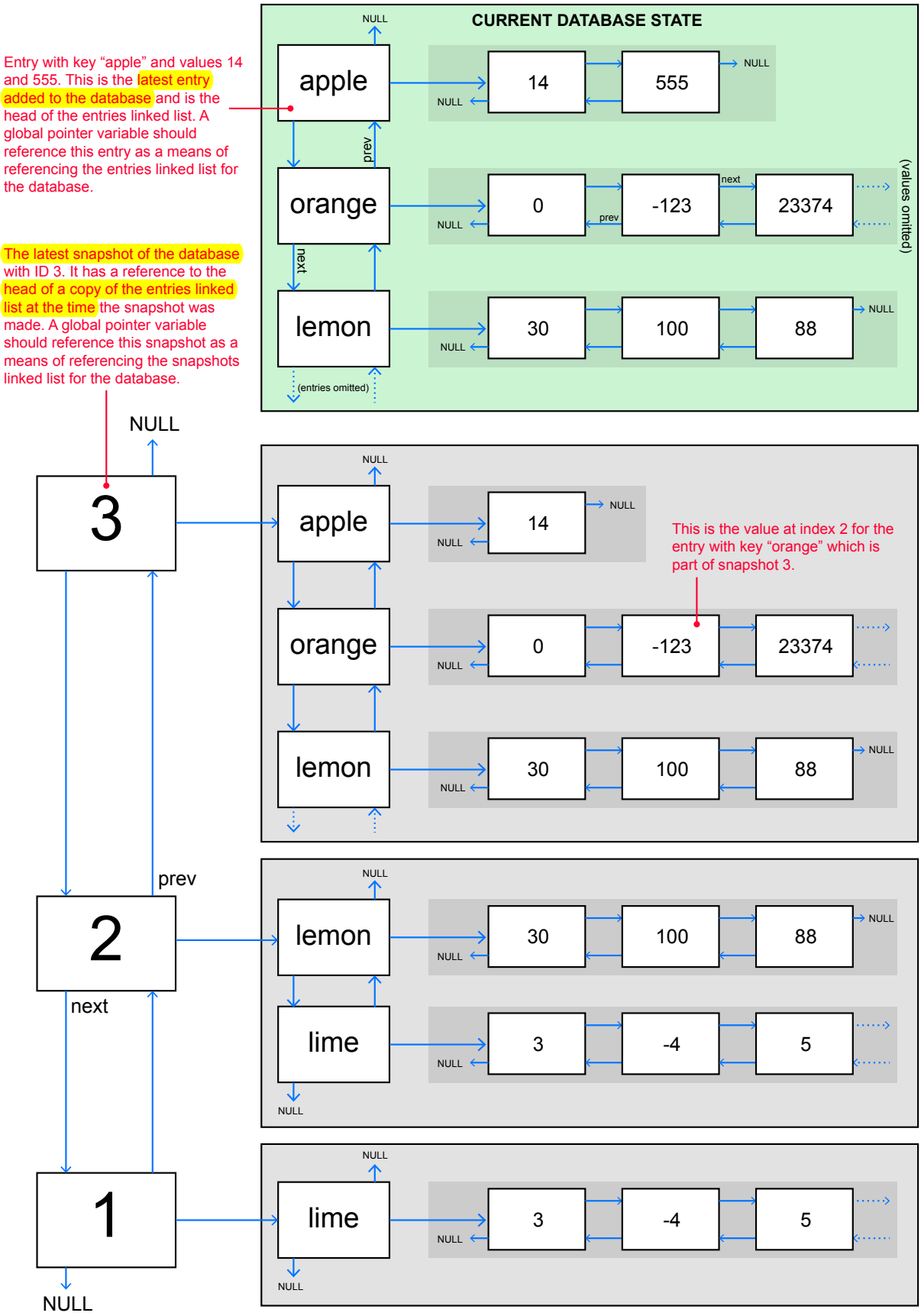
Your program must be contained in `snapshot.c` and `snapshot.h` and produce no errors when built and run on the lab machines and ed. Reading from standard input and writing to standard output.

Your program output must match the exact output format shown in the examples and on ed. You are encouraged to submit your assignment while you are working on it, so you can obtain some feedback.

The contents of an example header file `snapshot.h` are shown below

```
#ifndef SNAPSHOT_H
#define SNAPSHOT_H

#define MAX_KEY_LENGTH 16
#define MAX_LINE_LENGTH 1024

typedef struct value value;
typedef struct entry entry;
typedef struct snapshot snapshot;

struct value {
  value* prev;
  value* next;
  int value;
};

struct entry {
  entry* prev;
  entry* next;
  value* values;
  char key[MAX_KEY_LENGTH];
};

struct snapshot {
  snapshot* prev;
  snapshot* next;
  entry* entries;
  int id;
};

#endif
```

In order to obtain full marks, your program must free all of the dynamic memory it allocates. This will be automatically checked using `valgrind`. If your program produces the correct output for a given test case and does not free all memory it allocates, then it will not pass the given test case.


CURRENT DATABASE STATE
Entry with key “apple” and values 14 and 555. This is the latest entry added to the database and is the head of the entries linked list. A global pointer variable should reference this entry as a means of referencing the entries linked list for the database.
The latest snapshot of the database with ID 3. It has a reference to the head of a copy of the entries linked list at the time the snapshot was made. A global pointer variable should reference this snapshot as a means of referencing the snapshots linked list for the database.
apple
orange
lemon
lime
14
555
0
-123
23374
30
100
88
3
-4
5
(values omitted)
(entries omitted)
NULL
prev
next
3
2
1
This is the value at index 2 for the entry with key “orange” which is part of snapshot 3.

Commands

Your program should implement the following commands, look at the examples to see how they work.

- If a `<key>` does not exist in the current state, output: `no such key`
- If a `<snapshot>` does not exist in the database, output: `no such snapshot`
- If an `<index>` does not exist in an entry, output: `index out of range`

```
> HELP
BYE   clear database and exit
HELP  display this help message

LIST KEYS       displays all keys in current state
LIST ENTRIES    displays all entries in current state
LIST SNAPSHOTS  displays all snapshots in the database

GET <key>    displays entry values
DEL <key>    deletes entry from current state
PURGE <key>  deletes entry from current state and snapshots

SET <key> <value ...>     sets entry values
PUSH <key> <value ...>    pushes each value to the front one at a time
APPEND <key> <value ...>  append each value to the back one at a time

PICK <key> <index>   displays entry value at index
PLUCK <key> <index>  displays and removes entry value at index
POP <key>            displays and removes the front entry value

DROP <id>      deletes snapshot
ROLLBACK <id>  restores to snapshot and deletes newer snapshots
CHECKOUT <id>  replaces current state with a copy of snapshot
SNAPSHOT       saves the current state as a snapshot

MIN <key>  displays minimum entry value
MAX <key>  displays maximum entry value
SUM <key>  displays sum of entry values
LEN <key>  displays number of entry values

REV <key>   reverses order of entry values
UNIQ <key>  removes repeated adjacent entry values
SORT <key>  sorts entry values in ascending order

> BYE
bye
```

Examples

```
> LIST KEYS
no keys

> LIST ENTRIES
no entries

> LIST SNAPSHOTS
no snapshots

> SET a 1
ok

> GET a
[1]

> POP a
1

> GET a
[]

> POP a
nil

> PUSH a 2 1
ok

> GET a
[1 2]

> APPEND a 3 4
ok

> GET a
[1 2 3 4]

> DEL a
ok

> DEL a
no such key

> BYE
bye
```

```
> SET a 1
ok

> SET b 2 3
ok

> LIST KEYS
a
b

> LIST ENTRIES
b [2 3]
a [1]

> LIST SNAPSHOSTS
no snapshots

> PICK a 0
index out of range

> PICK b 1
2

> GET b 1
[2 3]

> PLUCK b 2
3

> GET b
[2]

> DEL b
ok

> GET b
no such key

> PURGE b
ok

> BYE
bye
```

```
> DROP 1
no such snapshot

> ROLLBACK 1
no such snapshot

> SET a 1 2
ok

> SET b 3 4
ok

> LIST ENTRIES
b [3 4]
a [1 2]

> SNAPSHOT
saved as snapshot 1

> SET c 5 6
ok

> LIST ENTRIES
c [5 6]
b [3 4]
a [1 2]

> SNAPSHOT
saved as snapshot 2

> PURGE b
ok

> ROLLBACK 1
ok

> CHECKOUT 2
no such snapshot

> LIST ENTRIES
a [1 2]

> LIST SNAPSHOTS
1

> BYE
bye
```

```
> SET a 1 4 2 3 4 2
ok

> SNAPSHOT
saved as snapshot 1

> MIN a
1

> MAX a
4

> SUM a
16

> LEN a
6

> REV a
ok

> GET a
[2 4 3 2 4 1]

> SORT a
ok

> GET a
[1 2 2 3 4 4]

> UNIQ a
ok

> GET a
[1 2 3 4]

> CHECKOUT 1
ok

> LIST SNAPSHOTS
1

> LIST ENTRIES
a [1 4 2 3 4 2]

> BYE
bye
```

Writing your own testcases

We have provided you with some test cases but these do not not test all the functionality described in the assignment. It is important that you thoroughly test your code by writing your own test cases.

You should place all of your test cases in the `tests/` directory. Ensure that each test case has the `.in` input file along with a corresponding `.out` output file. We recommend that the names of your test cases are descriptive so that you know what each is testing, e.g. `get-set.in` and `sort-uniq.in`

Submission details

Your attendance in the week 9 tutorial is required for the manual marking. Failure to attend your assigned tutorial will result in zero marks for this component, unless you get special consideration.

You must submit your code in the assignment page on ed. To submit, simply place your code into the code editor, click the run button to check your program works and then click the submit button.

You are encouraged to submit multiple times, but only your last submission will be marked.

Marking

In this assignment only valid inputs are tested, and your program will be checked for memory leaks and errors. If your program leaks memory you will not pass the test case even if the output is correct. In addition, we will run your program against a substantial collection of hidden test cases.

10 **marks** are assigned based on automatic tests for the *correctness* of your program.
This component will use hidden test cases that cover every aspect of the specification.

2 **marks** are assigned based on a manual inspection of the style and quality of your code.

Your program will be marked automatically, so make sure that you carefully follow the assignment specifications. Your program must match the exact output in the examples and the test cases on ed.

Warning: Any attempts to deceive or disrupt the marking system will result in an immediate zero for the entire assignment. Negative marks can be assigned if you do not properly follow the assignment specification, or your code is unnecessarily or deliberately obfuscated.