

COMP2129

Assignment 2

Due: 9:00am Monday, 18 April 2016

This assignment is worth 8% 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. It is important that you continually back up your assignment files onto your own machine, external drives, and in the cloud.

You are encouraged to submit your assignment on [Ed](#) while you are in the process of completing it. By submitting you will obtain some feedback of your progress on the sample test cases provided.

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 16
#define MAX_LINE 1024

typedef struct entry entry;
typedef struct snapshot snapshot;

struct entry {
    char key[MAX_KEY];
    int* values;
    size_t length;
    entry* next;
    entry* prev;
};

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

#endif
```

In order to obtain full marks, your program must free all of the dynamic memory it allocates. This will be automatically checked using address sanitizer. 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.

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

```
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 values to the front
APPEND <key> <value ...> appends values to the back

PICK <key> <index>   displays value at index
PLUCK <key> <index>  displays and removes value at index
POP <key>            displays and removes the front 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 value
MAX <key>         displays maximum value
SUM <key>         displays sum of values
LEN <key>         displays number of values

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

DIFF <key> <key ...> displays set difference of values in keys
INTER <key> <key ...> displays set intersection of values in keys
UNION <key> <key ...> displays set union of values in keys

> 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
b
a

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

> LIST SNAPSHOTS
no snapshots

> PICK a 0
index out of range

> PICK b 1
2

> GET b
[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 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 7 tutorial is required for the manual marking component.

Final deliverable for the correctness will be due in week 8. Further details will be announced on Ed.

You must submit your code and tests using the assignment page on [Ed](#). To submit, simply place your files and folders into the workspace, click run to check your program works and then click submit.

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 mark your program against a substantial collection of hidden test cases.

6 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 and tests.

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.