TU München, Fakultät für Informatik
Lehrstuhl III: Datenbanksysteme
Prof. Dr. Thomas Neumann

Exercises for *Foundations in Data Engineering*, WiSe 23/24

Alexander Beischl, Maximilian Reif (i3fde@in.tum.de)
http://db.in.tum.de/teaching/ws2324/foundationsde

Sheet Nr. 03

Exercise 1

Show off how well you can handle bash, pipes and gnu tools by now!

Build a persistent key-value store using only these tools. It should support two actions: **set** and **get**.

Set takes a key and a value and stores it.

Get retrieves the value stored for a given key.

You may assume that keys do not contain a `,` (comma). Also, it is not necessary to achieve $O(1)$ runtime for set and get as this task is about finding a very short solution. The functions can each be implemented in one line.

Once the implementation is done, do simple performance measurements:

- How long does it take to insert 1000 keys?
- How long does it take to retrieve 1000 keys afterwards?

Exercise 2

On Linux, a plethora of useful information can be found in the system *manual pages (man pages)*. They are especially relevant to a systems programmer, who often needs to interact with the kernel through system calls.

While man pages can of course also be found on the internet, we prefer to use the `man` executable to view man pages offline. That way, we can be sure that we are viewing the version of a man page that matches the programs installed on our system.

1. `man` does not display the text of the man page itself but it uses a helper program. Explain how `man` decides which helper program to use (Hint: Read the man page of `man`) in one sentence. How is the program called that is used on your system? Briefly describe your steps for finding the answer (1–2 sentences). List all commands you use.

Most systems usually use the program `less` to display man pages (although your system might be different). As man pages can be rather large, it is essential to be able to quickly search through man pages for relevant information.

2. Check the man page of `less`. How can you quickly search for all occurrences of a keyword (name the command)? How can you jump to specific lines in a file (name the command)? Briefly describe your steps for finding the answer (1–2 sentences).

Exercise 3

The `proc` filesystem (also called `procfs`) provides an interface to kernel data structures which can be queried for information about your system.

1. Use your knowledge about man pages and find a way to obtain information about your CPU with the `procfs`. Answer the following questions: Which file in the `procfs` contains the information about the available CPUs? Which helper program can be used to display the CPU infos in a nicer format?

In particular, the kernel exposes CPU flags which may (e.g., on ARM) influence the behavior of the `g++` option `-march=native` which was briefly introduced in the lecture. This option potentially sets a large number of other `-m<option>` options specific to the current CPU.

2. Check the `g++` man page for information about the various `--help` options. Use this information to answer the following question: Which `-m<option>` options does `g++` set on your system when using `-march=native`? Briefly describe your steps for finding the answer (2–3 sentences, also list all commands that you used).
3. Select 5 options that `g++` sets on your system when using `-march=native` from the last question. Briefly summarize their effect on the program that is produced by `g++` (one sentence per option).
4. If portability is an issue, it is usually not desirable to blindly apply the `-march=native` option during compilation. Explain why this is the case, and briefly outline a more flexible approach that leverages a build system (3–5 sentences).

Exercise 4

In the lecture, we introduced the C++ reference documentation at:

`https://en.cppreference.com/w/cpp`

This documentation should be the first place to go for any questions regarding the C++ language or standard library. Thus, being able to navigate and understand the reference documentation is an essential skill for a C++ programmer.

1. In the lecture, we introduced the value categories *lvalues* and *rvalues* but noted that this classification is inaccurate. Consult the reference documentation about value categories, and list the five value categories that C++ actually knows (no further explanation required).
2. Consider the following code fragment.

```
for (unsigned i = 0, j = 0; i < 10; ++i, j *= 2) {
    /* do something */
}
```

   Consult the reference documentation on `for` loops, and name the *syntactic* classification (e.g. statement, expression, ...) of the code fragments `unsigned i = 0, j = 0;` and `++i, j *= 2` in one sentence.

   In one of these code fragments, the comma (`,`) is an operator. Explain in which fragment this is the case and briefly outline the semantics of the comma operator (3–5 sentences).

3. Check the reference documentation on *implicit conversions* and briefly explain the key difference between numeric promotion and numeric conversion (3–5 sentences). Give two examples for each of these two conversions.

4. Check the reference documentation on *I/O manipulators* and answer the following questions with a short example (1 sentence each):
    - How can we print hexadecimal numbers?
    - How can we print numbers with a fixed width and leading zeros?
    - How can we right align numbers with a fixed width?