Lab 3 - ADTs, Recursion

CS 251, Fall 2024

Copyright Notice

Learning Goals

By the end of this lab, you'll:

- Select appropriate ADTs for a given situation
- Understand how to trace recursive function calls
- Write programs that use C++ STL implementations of common ADTs

Starter Code

lab03-starter.zip

Tasks

As usual, you can work on the tasks in any order. We strongly encourage collaboration in labs. Work with your peers, and ask your TAs questions!

(1) ADT Selection

Recall the 5 ADTs that we discussed in lecture:

- Vector or List
- Stack
- Queue
- Set
- Map

For each of the following scenarios, **which ADT would be most appropriate**? Briefly justify your answer and explain how the ADT would be used.

1. You're writing a website to handle ticket sales for popular events, such as concerts or conventions. To avoid overloading your server, you want to make sure that only a few

users are trying to check out at once. This means that you'll need to make some users wait for their turn to buy tickets.
2. Continuing with ticket sales, each ticket is for a specific seat/area. The ticket checkers and ushers want to be able to find information about a specific ticket.
3. In the lead-up to the ticket sale date, people will check the website frequently. You'd like to keep a record of all the website accesses and views.
4. The same people will visit the website multiple times. What unique visitors does the website have?
5. You quit your job writing ticket sale websites to write a text editor, and are implementing the "undo" operation. (Extra: how would you support "redo", or undo-ing an undo?)

(2) Recursion Tracing

This is a recursive function that reverses a string.

```
string reverseOf(string s) {
  if (s.empty()) {
    return "";
  } else {
    string ret = reverseOf(s.substr(1));
    return ret + s[0];
  }
}
```

Trace the execution of this function when called as `reverseOf("pots")`. At each level of recursion, what is the argument; what is `ret`; and what is returned?

(3) Map Reversal

One thing that we might want to do is "reverse" a map: given a value, get the key. Consider this possible function signature that, when called on a map, produces the reverse map: where each value is associated with its key. For example, the map `{"key" → "value"}` would be reversed to the map `{"value" → "key"}`.

```
map<string, string> reverseMap(map<string, string> m)
```

Both the **r**eturn type and the argument type can be improved. How would you change them?

Update the function declaration in `map_reverse.cpp`, implement it, and demonstrate that it works by updating the example in `main` and printing the result. Since you're changing

the function signature, we do not have automated tests for this function, but you can run your `main` with `make run_map_reverse`.

(4) Anagrams

An anagram is a word made by rearranging the letters of another word. For example:

- `meat` anagrams to `mate`, `meat`, `tame`, and `team`.
    - It does *not* anagram to `matte`. Even though both contain `m`, `e`, `a`, and `t`, the number of `t`'s is different.
- `tame` anagrams to `mate`, `meat`, `tame`, and `team`.
- `listen` anagrams to `listen`, `silent`, and `tinsel`.
- `magikarp` has no anagrams.

Note that a word is its own anagram.

Implement the `findAnagrams` function, which takes in a word and a dictionary, and returns a set of the anagrams of the word that are in the dictionary.

You should use a **specific abstract data type** to solve this problem! This function should iterate over the dictionary exactly once, and may not use recursion.

Use the commands `make run_anagrams` to run your program's `main` (not required, but useful for manual testing), and `make test_anagrams` to run the test suite.

Deliverables

To record your attendance at the lab, show a TA your group's answers and tests. You will be asked questions about your work!

TAs will discuss solutions in the last 15 minutes of the lab. Once you finish, you can work on the project until then. If you aren't interested in solutions, feel free to leave early once you've been checked off.

Acknowledgements

Content from Adam Blank, Sean Szumlanski et. al, revised and curated by Ethan Ordentlich