Tutorial 10

- Linked data structures!
Linked Data Structures – Overview

- In linked data structures, each element has a link to one (or more) other elements.

- Common linked data structures include
  - Linked Lists: each element has a link to the next element.
  - Doubly-linked lists: each element has a link to the next and the previous element.
  - Trees: each element has a link to $n$ children.†
  - Graphs: each element has a link to $n$ neighbours.†

† Simplified; please recall CS 135 for full definitions.
Linked Data Structures – C-specific

- In C, this link is implemented through pointers.
- Maintaining these pointers is one of the pitfalls when implementing linked data structures!
- Remember that you oftentimes have to distinguish between manipulating the first element (e.g., `tree->root` or `llist->front`), manipulating the last element (e.g., leaves or `llist->back`), and manipulating an element in between.
- Use diagrams and drawings *excessively* when planning on how to manipulate linked data structures!
Exercise - Removing from Back of Deque

Implement the following function:

```cpp
// deque_remove_back(deq) removes and returns the string that is stored at
// the back of the deque deq, or NULL if deq is empty. The client must free
// the returned string.
// effects: modifies deq
// time: O(1)
```

Hints:

- We diagrams and drawings **excessively** when planning on how
to manipulate linked data structures!
Exercise – An arbitrary record storing system

Implement the following functions:

// record_create(data, serial, data_clone,
// data_destroy, dat_print) creates
// a record storing data and serial, using the following
// function pointers:
// * data_clone can clone data;
// * data_dest can destroy data;
// * data_print can print data.
// effects: allocated heap memory; client must call
// record_destroy
// time: O(cl), where cl: time of data_clone
Exercise – An arbitrary record storing system cont.

Implement the following functions:

// record_destroy(rec) releases all resources used by rec
// effects: frees heap memory; rec becomes invalid
// time: \(O(\text{de})\), where \(\text{de}\): time of data_destroy

// record_print(rec) prints the content of rec to the console.
// effects: writes to console
// time: \(O(\text{pr})\), where \(\text{pr}\): time of data_print

// record_equals(rec1, rec2) returns true if rec1 is equal to rec2, and false otherwise.
// time: \(O(\text{eq})\), where \(\text{eq}\): time of data_equals