Today

- Exam 1 Overview
- Linked List wrap up
- Searching basics

Assignments

- HW4 out (due Tues)
- Exam 1 next Thurs
Exam 1 Overview

Basics

- Closed book, notes, etc.
- 10% of final grade
- 4 multipart questions

Possible Topics:

- everything from lectures and homework fair game
- class access modifiers (public vs protected vs private)
- subclassing and inheritance (including static vs dynamic binding and virtual functions)
- abstract classes (pure virtual functions)
- operator overloading
- the vector class (basics)
- templates (parametric types)
- pointers and linked lists
- binary search basics (today and tuesday)
- there will be some basic coding questions
Review: Implementing the rule of 3 functions for HW4

```cpp
template<typename K, typename V>
class LinkedListCollection : public Collection<K,V>
{
public:
...

// copy a linked list
LinkedListCollection(const LinkedListCollection<K,V>& rhs);
// assign a linked list
LinkedListCollection<K,V>& operator=(const LinkedListCollection<K,V>& rhs);
// delete a linked list
~LinkedListCollection();
...

private:
  struct Node { // stores key-value pair in node
    K key;
    V value;
    Node* next;
  };
  Node* head; // head pointer
  Node* tail; // tail pointer
  int length; // number of linked list nodes
};
...
```

Why implement all 3 of these for a linked list?

- if we leave them out, C++ provides default implementations
- and the default implementations introduce issues ...

The default copy constructor ...

- just copies member variables (e.g., this->head = rhs.head;)
- resulting in shared internal node structures
- changing one collection changes the other!
The default assignment operator ... 

- copies member variables as well
- leading to memory leaks (don’t delete lhs nodes)
- and shared internal node structures
Assignment operator hints:

Q: Why the return type in an assignment operator? (i.e., explain the signature)

```cpp
LinkedList& LinkedList::operator=(const LinkedList& rhs) {
    ...
}
```

Allows = to be chained together ... e.g., sometimes you’ll see:

```cpp
while ((p = p->next) != nullptr)
{
    ...
}
```

Q: What should happen when we do the following?

```cpp
LinkedList c1;
...
cl = c1;    // bad things happen here if not careful!
```

Q: How do we prevent bad things from happening in this case?

```cpp
LinkedList& LinkedList::operator=(const LinkedList& rhs) {
    if (this != &rhs) {
        // delete lhs nodes
        // copy rhs nodes to lhs
    }
    return *this;
}
```
Getting started with the gdb debugger ...

A debugger can help track down various code errors like segmentation faults

- **gdb** is a standard command-line debugging tool
- we’ll go over some basics, which might help you find some pointer issues

1. When compiling via `g++` use the `-g` flag
   - includes debug info / symbols
   - the HW4 CMakeLists.txt file includes this for you

2. Once you’ve created an executable, run **gdb** to start the debugger:
   
   ```
gdb hw4test
   ```

3. Open the code viewer within **gdb**
   
   ```
   (gdb) layout next
   ```
   - alternatively, type Ctrl-x Ctrl-a
   - should say “No source available” until you run your program ...

4. Run your program
   
   ```
   (gdb) run
   ```

5. **gdb** will stop where the error occurs
Some other useful `gdb` commands:

- **break** sets a breakpoint (e.g., `break main`)
- **step** steps through statements (lines)
- **run rand-10k.txt** runs with an input file
- **p var** prints the value of `var`