Today
  • Quiz 3
  • Linked List Review

Assignments
  • HW3 due
  • HW4 out (soon)
Linked Lists

Linked lists consist of “chained together” nodes

```c
struct Node {
    K key;
    V value;
    Node* next;
};
```

We create new nodes via dynamic allocation

```c
define head = new Node; // dynamically allocate space
head->key = k1; // assign the key attribute
head->value = v1; // assign the value attribute
head->next = nullptr; // instead of NULL
```

Nodes are “linked” together via next pointers

```c
head->next = new Node; // Alt: Node* ptr = head->next;
head->next->key = k2; // ptr->key = k2;
head->next->value = v2; // ptr->value = v2;
head->next->next = nullptr; // ptr->next = nullptr;
```

In HW-4 we use both a head and tail pointer ...

---

Q: Why do we use both? ... to quickly insert at end of list
Exercise 3
```cpp
#ifndef LINKED_LIST_COLLECTION_H
#define LINKED_LIST_COLLECTION_H

#include <vector>
#include <algorithm>
#include "collection.h"

template<typename K, typename V>
class LinkedListCollection : public Collection<K,V>
{
public:
    // create an empty linked list
    LinkedListCollection();
    // 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();

    // collection functions:
    void add(const K& a_key, const V& a_val);
    void remove(const K& a_key);
    bool find(const K& search_key, V& the_val) const;
    void find(const K& k1, const K& k2, std::vector<V>& vals) const;
    void keys(std::vector<K>& all_keys) const;
    void sort(std::vector<K>& all_keys_sorted) const;
    int size() const;

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
};

... function implementations ...

#endif
```
Q: What is the C++ "rule of three"?

- A class that defines a destructor, assignment operator, or a copy constructor should define all three.

Q: For a linked list, what is the “algorithm” for the destructor?

- In particular, draw step-by-step what should happen on a 3-node list.
- Useful to write a helper function to “make_empty()” (see below).

Q: For a linked list, what is the “algorithm” for the copy constructor?

- Note that a new object is created (so, e.g., nothing to delete).
- General idea: insert copy of each rhs node into lhs (e.g., via insert()).

Q: For a linked list, what is the “algorithm” for the assignment operator?

- Note we are assigning into an existing object (with nodes)!
- First: delete all of the lhs nodes (e.g., via make_empty()).
- Then: copy rhs nodes into lhs.
- But have to be careful with first step! ... more soon.

S. Bowers 5 of 6
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;
...
c1 = 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;
}
```