Big 5 Continued and Lists

Load Testing UA

Case Study: StringHolder

```
class StringHolder {
     std::string *string_;
public:
     StringHolder() : string_(nullptr) {};
     StringHolder(const char *initial_string);

     const char* c_str() const noexcept;
     void ChangeString(const char *input_string);
};
```

Rule of Three - Destructor

```
~StringHolder(){
     delete string_;
}
```

Rule of Three - Copy Constructor

```
StringHolder(const StringHolder &source) {
     if(source.string_) {
          string_ = new std::string(*source.string_);
     } else {
          string_ = nullptr;
     }
}
```

Rule of Three – Copy Assignment Operator

```
StringHolder& operator=(const StringHolder &source){
	if(this == &source) {
		return;
	}

	delete string_;
	string_ = nullptr;
	if(source.string_) {
		string_ = new std::string(*source.string_);
	}
	return *this;
}
```

Rule of Three vs Rule of Five

- Rule of five (move semantics new with C++11)
  - Move Constructor
  - Move Assignment Operator

- rvalue reference &&
  - Can bind to a temporary (rvalue)

Rule of Five - Move Constructor

```
StringHolder(StringHolder&& source) {
     string_ = source.string_;
     source.string_ = nullptr;
}
```

Rule of Five – Move Assignment Operator

```
StringHolder& operator=(StringHolder&& source) {
     if(this == &source) {
          return;
     }

     delete string_;
     string_ = source.string_;
     source.string_ = nullptr;
     return *this;
}
```

Linked Lists

Linked List Data Structures

```
struct ListNode {
      Data data_;
      ListNode next_;
      ListNode(Data d) : data_(d), next_(nullptr) {};
}

class LinkedList {
      ListNode head_;
      …
public:
      …
```

Insert at start

```
void InsertHead(Data new_data) {
     ListNode *new_node = new ListNode(new_data);
     new_node->next_ = head_;
     head_ = new_node;
}
```

Remove Head

```
void RemoveHead() {
     if(!head_) { return; }
     ListNode *tmp = head_;
     head_ = head_->next_;
     delete tmp;
}
```

Remove Tail

```
void RemoveTail() {
     if(!head_) { return; }
     ListNode *remove_next = head_;
     while(remove_next->next_ &&
                remove_next->next_->next_) {
           remove_next = remove_next->next_;
     }
     delete remove_next->next_;
     remove_next = nullptr;
}
```

VisuAlgo

https://visualgo.net/en/list