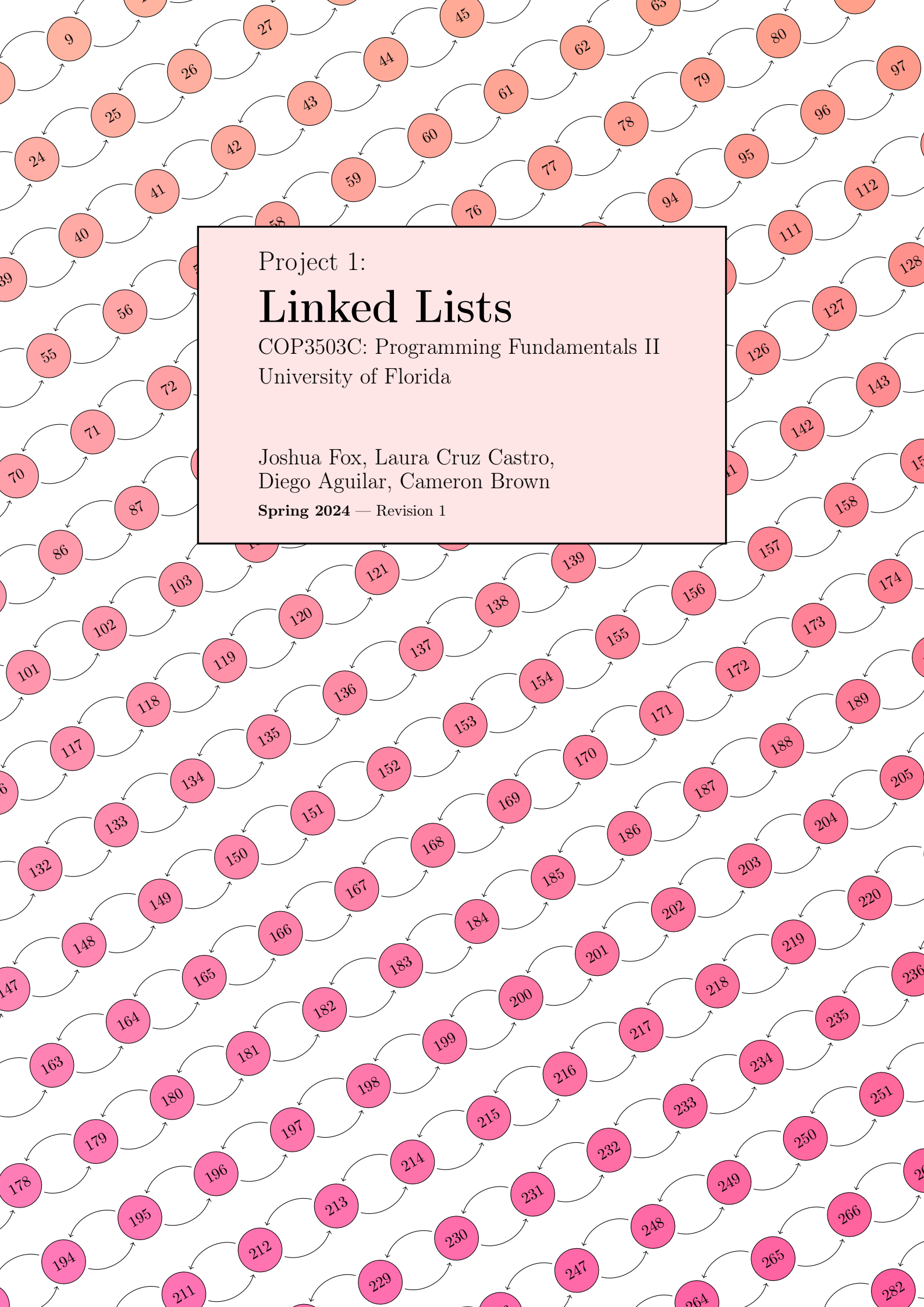

Project 1:

Linked Lists

COP3503C: Programming Fundamentals II
University of Florida

Joshua Fox, Laura Cruz Castro,
Diego Aguilar, Cameron Brown

Spring 2024 — Revision 1

1 Code Structure

As shown above, the Linked List class itself stores very little data: Pointers to the first and last nodes, and a count. In some implementations, you might only have a pointer to the first node, and that's it. In addition to those data members, your Linked List class must conform to the following specification! (Yes it's a lot... it's a big project get started early!)

1.1 Construction / Destruction

Default Constructor	Default constructor. How many nodes in an empty list? (Answer: 0) What is head pointing to? What is tail pointing to? (Answer: nullptr) Initialize your variables!
Copy Constructor	Sets "this" to a copy of the passed in LinkedList. For example, if the other list has 10 nodes, with values of 1-10? "this" should have a copy of that same data.
Destructor	The usual. Clean up your mess. (Delete all the nodes created by the list.)

1.2 Behaviors

PrintForward	Iterator through all of the nodes and print out their values, one a time.
PrintReverse	Exactly the same as PrintForward, except completely the opposite.
PrintForwardRecursive	This function takes in a pointer to a Node—a starting node. From that node, recursively visit each node that follows, in forward order, and print their values. This function **MUST** be implemented using recursion, or tests using it will be worth no points. Check your textbook for a reference on recursion.
PrintReverseRecursive	Same deal as PrintForwardRecursive, but in reverse.

1.3 Accessors

NodeCount	How many things are stored in this list?
FindAll	Find all nodes which match the passed in parameter value, and store a pointer to that node in the passed in vector. Use of a parameter like this (passing a something in by reference, and storing data for later use) is called an **output parameter**.
Find	Find the first node with a data value matching the passed in parameter, returning a pointer to that node. Returns **nullptr** if no matching node found.
GetNode	Given an index, return a pointer to the node at that index. Throws an exception of type **out_of_range** if the index is out of range. Const and non-const versions.
GetHead	Returns the head pointer. Const and non-const versions.
GetTail	Returns the tail pointer. Const and non-const versions.

1.4 Insertions

AddHead	Create a new Node at the front of the list to store the passed in parameter.
AddTail	Create a new Node at the end of the list to store the passed in parameter.
AddNodesHead	Given an array of values and its size, insert a node for each of those at the beginning list, maintaining the original order.
AddNodesTail	Ditto, except adding to the end of the list.
InsertAfter	Given a pointer to a node, create a new node to store the passed in value, after the indicated node.
InsertBefore	Ditto, except insert the new node before the indicated node.
InsertAt	Inserts a new Node to store the first parameter, at the index-th location. So if you specified 3 as the index, the new Node should have 3 Nodes before it. Throws an **out_of_range** exception if given an invalid index.

1.5 Removals

RemoveHead	Deletes the first Node in the list. Returns whether or not the Node was removed.
RemoveTail	Deletes the last Node, returning whether or not the operation was successful.
Remove	Remove ALL Nodes containing values matching that of the passed-in Returns how many instances were removed.
RemoveAt	Deletes the index-th Node from the list, returning whether or not the operation was successful.
Clear	Deletes all Nodes. Don't forget the node count—how after you deleted all of them?

1.6 Operators

operator[]	Overloaded subscript operator. Takes an index, and returns data from the indexth node. Throws an **out_of_range** exception for an invalid index. Const and nonconst versions.
operator=	Assignment operator. After listA = listB, listA == listB is true. Can of your existing functions to make write this one? (Hint: Yes you can.)
operator==	Overloaded equality operator. Given listA and listB, is listA equal to listB? What would make one Linked List equal to another? If each of its nodes were equal to the corresponding node of the other. (Similar to comparing two arrays, just with non-contiguous data).

2 Tips

A few tips for this assignment:

- Start small! Work on one bit of functionality at a time. Work on things like Add() and PrintForward() first, as well as accessors (brackets operator, Head()/Tail(), etc). You can't really test anything else unless those are working.
- Your output is simple: print the data, print newline
- Remember the "Big Three" or the "Rule of Three"

  - If you define one of the three special functions (copy constructor, assignment operator, or destructor), you should define the other two
- Refer back to the recommended chapters in your textbook as well as lecture videos for an explanation of the details of dynamic memory allocation

  - There are a lot of things to remember when dealing with memory allocation
- Make charts, diagrams, sketches of the problem. Memory is inherently difficult to visualize, find a way that works for you.
- Don't forget your node count!