ECE244 Lab 4: A Resistor Network Simulator

1 Objectives

This assignment will give you practice in creating and manipulating linked lists in C++. You will also see how to programs can be used to simulate hardware circuitry.

2 Problem Statement

The lab builds on Lab 3, but adds several new features.

- You will replace arrays with linked lists to eliminate the maximum-size limitations of the previous program, and to allow you to implement the `delete` *resistor* command.
- You will add a new `solve` capability to the program so that it can find the dc voltage at each node.

3 Preparation & Background

This lab builds on the previous lab in its use of classes, constructors, and member functions. You should review the textbook and lecture notes pertaining to the following topics:

- Constructors and destructors
- Class member variables and functions
- Public/private access types
- Dynamic memory management with new and delete
- Linked lists

4 Specifications

The program shall build on the program you developed in Lab 3, with the most siginificant modifications summarized in the following list. Detail is given in the appropriate subparts of Sec 4 below.

- The program shall accept any number of resistors, nodes, and resistors per node using linked lists
- The program will no longer require or accept the `maxVal` command, and also need not accept the `printR all` command
- The program will now accept the `delete` *resistor* command
- In the `printNode all` command, nodes containing no resistors shall not be printed
- Your program shall implement new commands to set the voltage of a node to a fixed value, and to solve for the voltages at other nodes.

4.1 Coding Requirements

1. The Standard Template Library (STL) shall not be used - the point of this assignment is for you to create your own linked list implementation
2. There shall be no predefined maximum number of resistors, nodes, or resistors per node
3. No global variables shall be used
4. Linked lists shall be used to replace the array of resistors, nodes, and resistors attached to each node
5. All class data member variables, including `Node` and `Resistor`, shall be of private access type
6. The corresponding `Resistor` object(s) shall be deleted (memory freed) when the `deleteR` command is entered
7. The program shall not leak memory

4.2 Input

The commands your program must accept, the output each generates if it successful, and the action to take are listed in Table 1. Commands that are new to this lab are in the top portion of the table, while those that are carried over from Lab 3 are in the lower part of the table.

For each valid line of input (i.e. each line not causing an error as defined in Section 4.3), one or more lines of output shall be produced as described in Table 1 and below. The values in italics must be replaced by either the value given in the command, or the value already stored (eg. *resistance_old* in the `modifyR` output defined in Table 1). Strings must be reproduced exactly as entered. An example session is provided in Sec 5 to illustrate this.

4.3 Error Checking

Input shall be checked for errors as described in Lab 3, except that the program shall no longer produce errors for "resistor array is full" or "node is full" because the linked list implementation has no set maximum size. There is also no longer an error for node values being out of range – any integer is now a valid node id (even a negative integer). Table 2 lists the errors for which your program must check. If more than one error occurs on a line of input, only one error message shall be issued: the first error message occuring as arguments are processed from left to right. If a single argument has more than one error, the one listed first in Table 2 should be printed.

4.4 Output

4.4.1 `printR` and `printNode` command output

Resistance information should be printed identically to Lab 3, except that the `printR all` command will not be used or tested in this lab. Node information should be printed in the same way as Lab 3, except that *nodes with no resistors should not be printed*. Nodes shall be printed in ascending order of node ID, and the resistors attached to a node shall be printed in the order in which they were added to the node.

Command	Arguments	Output if valid	Action if valid
setV	nodeid voltage	Set: node *nodeid* to *voltage* Volts	Updates the specified node voltage data member. This corresponds to connecting this node to a voltage source.
unsetV	nodeid	Unset: the solver will determine the voltage of node *nodeid*	Updates the specified node to mark its voltage as unknown (no longer connected to a voltage source).
solve		Solve: *node voltage info (see below)*	Runs a numerical solution technique to determine the voltage of all nodes that do not have their voltages set.
deleteR	name	Deleted: resistor *name*	The specified resistor is removed from the lists.
deleteR	`all`	Deleted: all resistors	All resistors deleted so we have an empty network
insertR	name resistance nodeid nodeid	Inserted: resistor *name resistance* Ohms *nodeid* -> *nodeid*	Adds a new resistor to the resistor linked lists in both nodes
modifyR	name resistance	Modified: resistor *name* from *resistance_old* Ohms to *resistance* Ohms	Updates the resistance of a resistor; note that this resistor must be updated in the linked lists within two nodes.
printR	name	Print: *resistor info (see below)*	No data changed
printNode	nodeid	Print: *node info (see below)*	No data changed
printNode	`all`	Print: *node info (see below)*	No data changed

Table 1: Valid commands and arguments

Error message	Cause
Error: no nodes have their voltage set	The `solve` command was run, but no nodes to which resistors are attached have a set (known) voltage. No solution can be found in this case.
Error: resistor *name* not found	When searching for a resistor by name (eg. in modifyR, printR, deleteR), a resistor with the given name was not found
Error: resistor *name* already exists	When adding a new resistor, the resistor name alread exists
Error: node *nodeid* not found	Whe setting/unsetting; a node with the given *nodeid* was not found

Table 2: Errors to be reported in this lab.

4.4.2 `solve` command

The solve command first determines the voltage of every node, and then it prints the voltage of every node in ascending node order. To find the voltage at every node in a network, we can follow the iterative procedure below.[1]

```
Initialize the voltage of all nodes without a specified (setV) voltage to 0.
while (some node's voltage has changed by more than MIN_ITERATION_CHANGE) {
          for (all nodes without a set voltage) {
                    set voltage of node according to Eq. 3
          }
}
```

The voltage of a node is computed from the voltages of its neighbours according to Kirchoff's current equation, which states that the total current entering or leaving a node must be 0. Consider $node_0$ below, which is surrounded is connected by 3 resistors to 3 other nodes, as shown in Fig. 1. The total current entering $node_0$ must be 0:

$$I_a + I_b + I_c = 0 \tag{1}$$

We can rewrite this using the fact that the current through each resistor is simply the voltage across it divided by its resistance.

$$\frac{V_1 - V_0}{R_a} + \frac{V_2 - V_0}{R_b} + \frac{V_3 - V_0}{R_c} = 0 \tag{2}$$

Rearranging and solving for the voltage at $node_0$, V_0, yields:

$$V_0 = \frac{1}{\frac{1}{R_a} + \frac{1}{R_b} + \frac{1}{R_c}} \times \left[\frac{V_1}{R_a} + \frac{V_2}{R_b} + \frac{V_3}{R_c}\right] \tag{3}$$

[1]The iterative procedure we are using to solve for the node voltages is called the Gauss-Seidel method. It is a very general method of solving linear systems of equations, and is used to solve circuits and other systems of equations with thousands or even millions of unknowns.

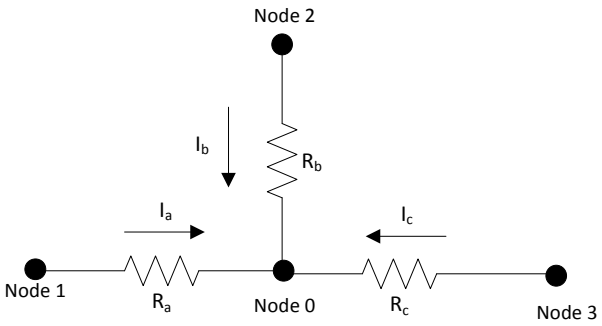

Figure 1: A portion of a resistor network. Node 0 is connected to three other nodes as shown.

The iterative procedure above determines what voltage a node must have for it to balance the current coming through all the resistors connected to it. Once we update the voltage of some node (say node *#0*) however, the voltage of other nodes (e.g. node *#1*) connected to that node may have to change to ensure the current flowing into them is in balance. Hence we iterate through all the nodes, repeatedly updating the voltages of those that do not have a fixed (`setV`) voltage according to Eq. 3 until no voltage changes much – at that point we have *converged* to a solution. For this lab, you should iterate until no node changes by more than `MIN_ITERATION_CHANGE`, which you should define to be 0.0001.

Once your solver has converged, print (to cout) the voltage of every node that has at least one resistor connected to it, in ascending node order. For the example shown in Fig 2, the solve command would print:

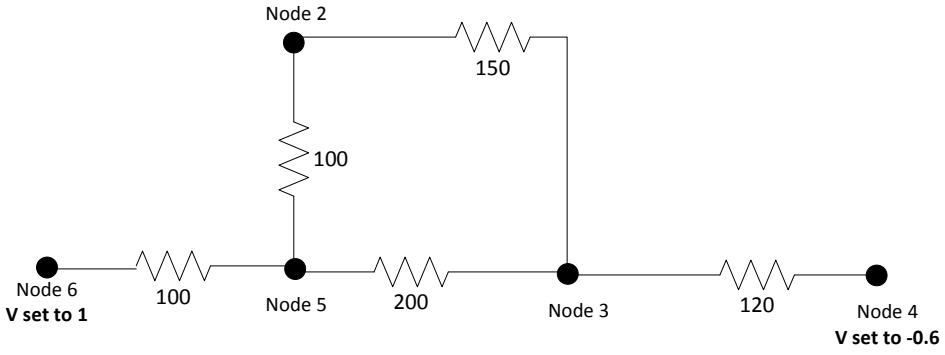

Figure 2: A more complex resistor network. Two nodes have been connected to voltage sources with `setV`, and our program will solve for the other 3 voltages.

```
Solve:
  Node 2: 0.30 V
```

```
  Node 3: -0.02 V
  Node 4: -0.60 V
  Node 5: 0.52 V
  Node 6: 1.00 V
```

5 Sample Session

```
> insertR R1 100 6 5
Inserted: resistor R1 100.00 Ohms 6 -> 5
> insertR R2 100 5 2
Inserted: resistor R2 100.00 Ohms 5 -> 2
> insertR R3 200 5 3
Inserted: resistor R3 200.00 Ohms 5 -> 3
> insertR R4 150 2 3
Inserted: resistor R4 150.00 Ohms 2 -> 3
> insertR R5 120 3 4
Inserted: resistor R5 120.00 Ohms 3 -> 4
> solve
Error: no nodes have their voltage set
> setV 6 1
Set: node 6 to 1.00 Volts
> setV 4 -0.6
Set: node 4 to -0.60 Volts
> solve
Solve:
  Node 2: 0.30 V
  Node 3: -0.02 V
  Node 4: -0.60 V
  Node 5: 0.52 V
  Node 6: 1.00 V
> unsetV 4
Unset: the solver will determine the voltage of node 4
> solve
Solve:
  Node 2: 1.00 V
  Node 3: 1.00 V
  Node 4: 1.00 V
  Node 5: 1.00 V
  Node 6: 1.00 V
> setV 4 -0.6
Set: node 4 to -0.60 Volts
> modifyR R5 1000
Modified: resistor R5 from 120.00 Ohms to 1000.00 Ohms
> solve
Solve:
  Node 2: 0.81 V
  Node 3: 0.72 V
  Node 4: -0.60 V
  Node 5: 0.87 V
  Node 6: 1.00 V
> printR R5
Print:
R5                    1000.00 Ohms 3 -> 4
> printNode 2
Print:
Connections at node 2: 2 resistor(s)
  R2                     100.00 Ohms 5 -> 2
```

```
  R4                     150.00 Ohms 2 -> 3
> deleteR R2
Deleted: resistor R2
> printNode all
Print:
Connections at node 2: 1 resistor(s)
  R4                     150.00 Ohms 2 -> 3
Connections at node 3: 3 resistor(s)
  R3                     200.00 Ohms 5 -> 3
  R4                     150.00 Ohms 2 -> 3
  R5                    1000.00 Ohms 3 -> 4
Connections at node 4: 1 resistor(s)
  R5                    1000.00 Ohms 3 -> 4
Connections at node 5: 2 resistor(s)
  R1                     100.00 Ohms 6 -> 5
  R3                     200.00 Ohms 5 -> 3
Connections at node 6: 1 resistor(s)
  R1                     100.00 Ohms 6 -> 5
> deleteR all
Deleted: all resistors
> printNode all
Print:
> unsetV 1000
Error: node 1000 not found
> solve
Error: no nodes have their voltage set
> insertR Rnew 100 -5000 3333
Inserted: resistor Rnew 100.00 Ohms -5000 -> 3333
> deleteR RINeverCreated
Error: resistor RINeverCreated not found
> insertR Rnew 240 2 4
Error: resistor Rnew already exists
>
```

6 Suggested Data Structure

This section discusses one data structure for your circuit that you may opt to use, diagrammed in Fig 3. The primary object representing the circuit is a `NodeList`. The `NodeList` holds a linked list of `Node`s, each of which contains a `ResistorList` holding `Resistor`s. For each resistor that is added to the circuit, two copies are created: one for each `Node` it connects to within the `NodeList`.

Functions that may be useful in `NodeList` include:

- "Find node": Accept a node ID and return a pointer to the corresponding `Node`, or `NULL` if it does not exist
- "Find or insert node": Accept a node ID and return a pointer to the corresponding `Node` if it exists, or create a new one
- Determine if a resistor with a given label exists in any of the `Node`s
- Change the resistance of a resistor by name (or return a failure indication)
- Delete a resistor by name (or return a failure indication)
- Delete all resistors in the list

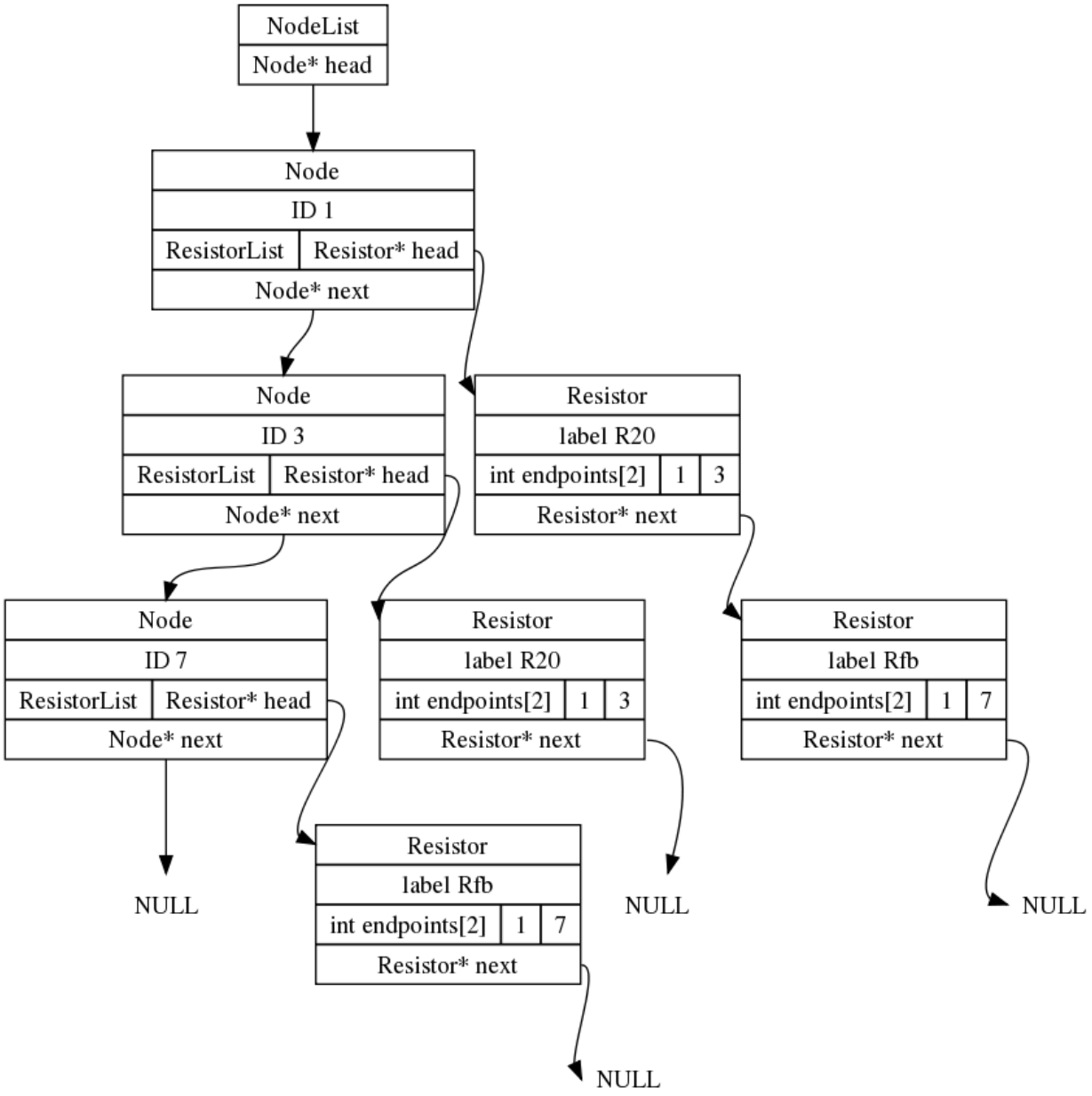

Figure 3: Suggested data structure. The state shown is what would exist after the commands `insertR R20 100 1 3` and `insertR Rfb 100 1 7`.

Likewise, functions that may be useful in `ResistorList` include:

- Insert a resistor at the end of the list
- Find a resistor by name, returning a pointer to it
- Delete a resistor given a pointer

7 Helpful Hints

- You should re-use your code from the previous lab to the extent possible. You will need to alter the data structures, and possibly make small output alterations.
- The global variables that hold the node and resistor arrays in Lab 3 should be deleted. Their functionality will now be provided by objects of class `NodeList` and `ResistorList`.
- To avoid use of global variables, you should create your `NodeList` object within `main()` and pass a reference to it (`NodeList&`) to any function that needs access (eg. your parser, add/change/remove commands, etc.)
- You will need new data members in class `Node` to store the voltage, and whether the node has a fixed voltage (`setV`) or not.
- Each block of memory allocated by `new` must be freed by `delete` *exactly once*.
- The linked list will start empty. You will need to create objects of class `Node` as you go (as they are referenced by resistors that are added). It may be useful to you to create `Node* NodeList::findOrInsert(int nodeID)` which finds or creates a node with the given ID.
- When using dynamic memory, think carefully if a class needs a destructor and what objects it might need to `delete` (hint: anything where the class holds the only pointer to a memory block allocated with `new`).
- Listing all nodes correctly will be made easier if you maintain your node list in ascending order of node ID.
- If you use the suggested data structure, remember when adding/changing resistors that you will have two copies of each resistor.
- Use of the `valgrind` memory tester is strongly recommended to catch invalid reads, writes, and allocation/deallocation. It will tell you when and on what function and line the error occurs. For memory leaks, it will tell you where the leaking block was allocated.
- There are several corner cases that, in principle, require special handling. However, you should ignore these cases and your code will NOT be tested against these cases.
  - It is possible to have nodes that are not connected to anything. You should ignore this case.
  - It is also possible to have two disjoint networks, one with voltage and one that is "floating". You should also ignore this case.
  - You may assume that no resistor will have a 0 value.
  -

8 Procedure

Create a `lab4` folder with appropriate permissions in your `ece244` directory. Before submitting, remember to use `exercise`, plus to make some test cases of your own as exercise will not test all cases.

9 Deliverables

You must submit all source files to permit your project to compile. They should be:

- Main.cpp
- Rparser.cpp and Rparser.h
- ResistorList.h and ResistorList.cpp
- Resistor.h and Resistor.cpp
- Node.h and Node.cpp
- NodeList.h and NodeList.cpp

Submit the files using
`~ece244i/public/submit` 4