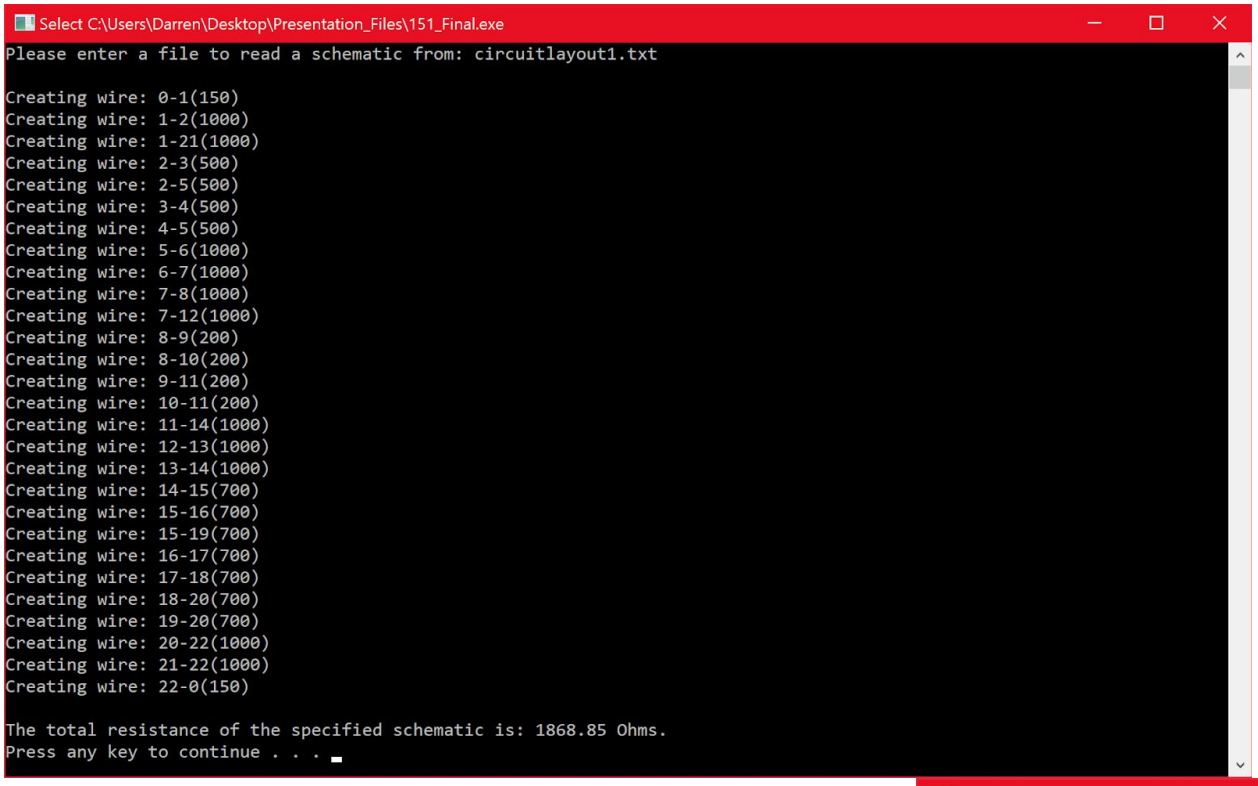
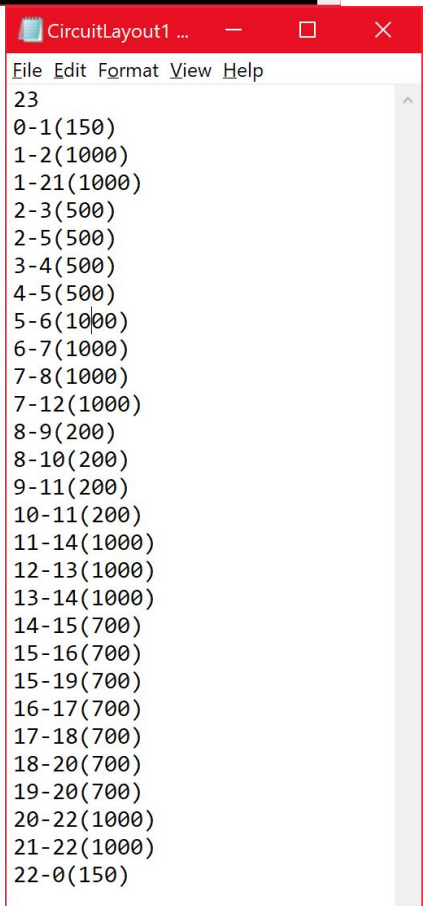
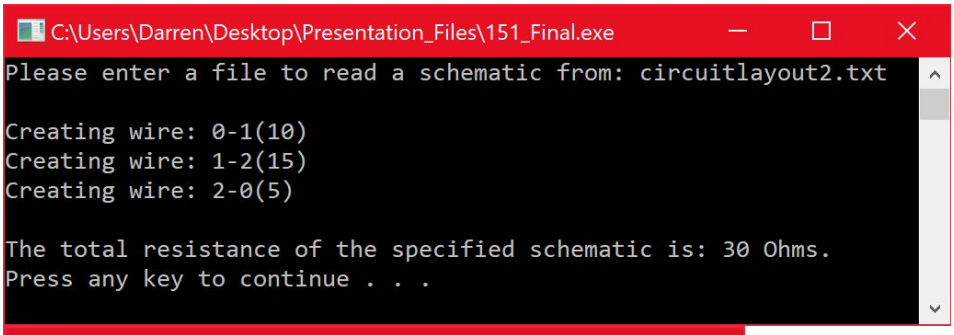
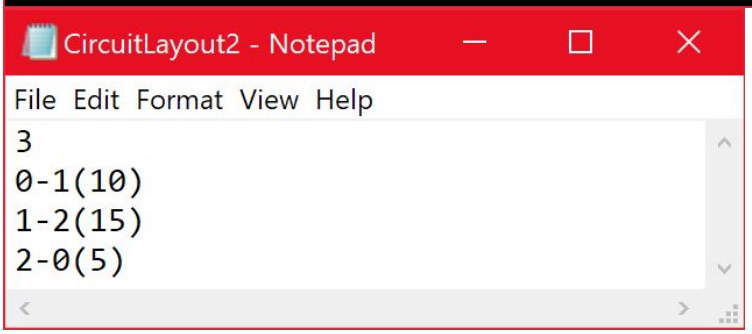
CS 151 Final Project

Program Output:

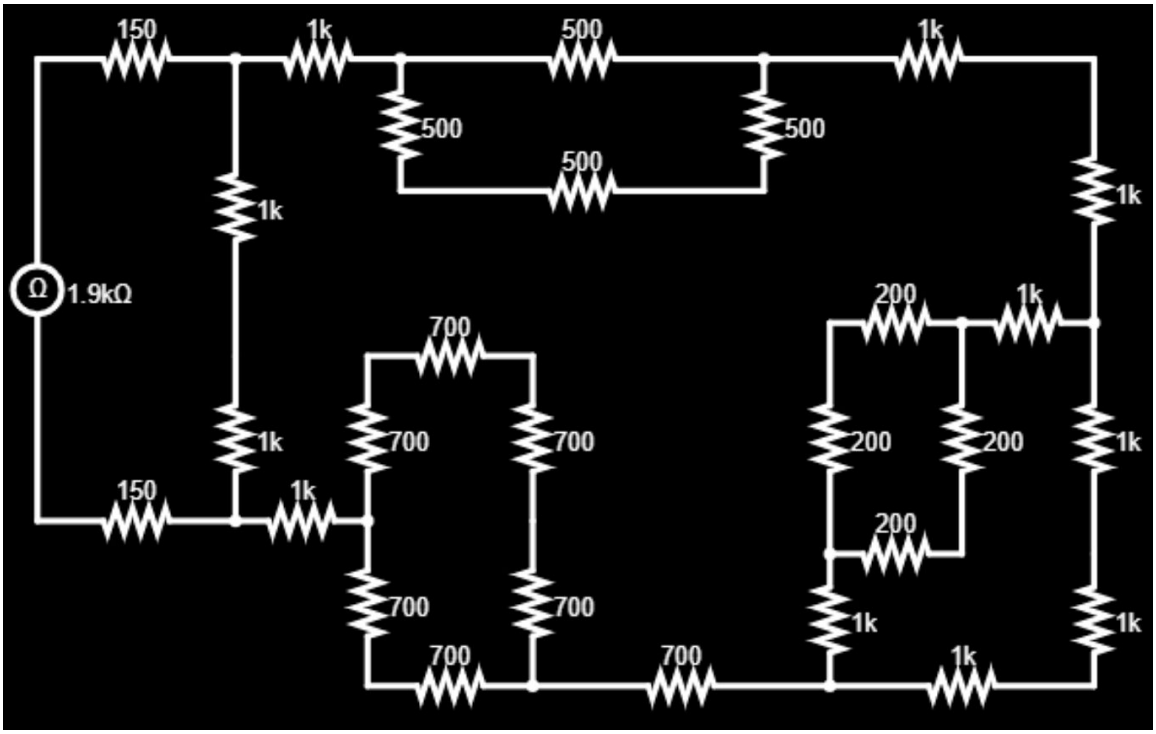








Circuit Modeled by CircuitLayout1.txt:



Program Code:

```
/*********************************************************************
 * Course:CS 151                Day & Time: Thursday 5:30-9:20
 *
 * Project: Final
 * Programmer: Darren Vawter
 * Date Created: 13MAY19
 *
 * Program Title: Circuit Analyzer
 * Program Description: This program creates a bi-nodal linked graph to represent the junctions
 *                      and wires of standard series/parallel resistive circuits. It reads in a
 *                      schematic description from a plain text file which can easily be
 *                      manipulated by a user in order to modify the circuit being analyzed. The
 *                      core algorithm of this circuit analyzer is the calcResistance() function
 *                      which indirectly calls two other recursive functions that traverse the
 *                      circuit network in order to calculate the total resistance of the circuit.
 *                      The implementation of this core algorithm is explained in further detail
 *                      in the algorithms section below.
 *
 * Algorithm:
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~main()~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 *      -prompt user for, and get, schematic file name
 *      -attempt to open the specified file name as an input stream, if failed, go back one step
 *      -get the first line from the file
 *              -use the value from the first line as the argument to initalize the Circuit object
 *      -get the next line from the file
 *      -loop while the input file has not thrown its fail flag
 *              -parse src int from first char to '-'
 *              -parse dest int from '-' to '('
 *              -parse resistance double from '(' to ')'
 *              -call the wireJunctions() method on the created Circuit objects
 *                      -pass the parsed src, dest, and resistance values as arguments
 *              -get the next line
 *      -close the input file
 *      -calc and display the circuit's total resistance (by calling calcResistance())
 *
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~calcResistance()~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 *      -call calcTotalResistance() on root junction (this is for encapsulation purposes)
 *
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~calcTotalResistance()~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 *      -get forward connections of relative root arg
 *      -if there is only 1 connection
 *              -if node on the other side of the connection is final node (0) //~~BASE CASE~~
 *                      -return the resistance of the connection
```

```
*                -else
*                        -return the resistance of the connection plus calcTotalResistance() of the
*                         node on the other side of the connection
*        -if there are multiple connections
*                -create an instance of CollisionResult and set it to the return of calcEqResistance
*                 of the relative root argument
*                -return the resistance of the collision result + calcTotalResistance() with the
*                 colliding node of the collision result as the argument for relative root
*
*~~~~~~~~~~~~~~~~~~~~~~~~~~calcTotalResistance()~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
*        -init total resistance to 0
*        -create arrays of resistances and Junction pointers to traverse multiple paths in parallel
*        -init path resistances all to 0 and Junction pointers to each of the relative root's forwards
*        -init collided to false
*        -loop while collided is false
*                -find the current node in the Junction pointers array with the lowest ID#
*                -look at the number of forward connections on that node
*                -if there is more than 1 forward connection
*                        -create an instance of CollisionResult and set it to the return of
                          calcEqResistance() of that node
*                        -add the resistance of the collision result to the position in the resistance
*                         array that is parallel to the position of the Junction with the min index
*                        -set the Junction pointer at that previous min index to the collision node
*                -if there is only 1 forward connection
*                        -add the resistance of the connection to the position in the resistance
*                         array that is parallel to the position of the Junction with the min index
*                        -set the Junction pointer at that previous min index to the next node up
*                -set collided to true
*                -loop through each Junction pointer in the array
*                        -if that Junction pointer @ i is not equal to the junction pointer @ 0
*                                -set collided to false
*                                -break
*                -if collided is true
*                        -loop through the resistance in the resistance array and sum the inverses
*                        -invert the sum
*                -create a collision result instance, set it's node equal to any Junction ptr in the
*                 array since they are all the same at this point
*                -set the resistance equal to the inverted sum that was just calculated
*                -delete[] the two dynamic arrays
*                -return the collision result object
*
*****************************************************************************/
```

Driver.cpp

```
#include <cstdlib>

#include <iostream>
#include <string>
#include <fstream>
#include "Circuit.h"

using namespace std;

int main() {

	//get schematic file from user
	string fileName;
	GETFILE://label to come back to
	cout << "Please enter a file to read a schematic from: ";
	getline(cin, fileName);

	//create input stream and begin looping through file
	ifstream schematic(fileName.c_str());

	//check if file opened
	if (!schematic.is_open()) {
		cout << "Unable to open file.\n\n";
		goto GETFILE;
	}

	//init vars to pull from file
	string parseln, ele;
	int src, dest, resistance;

	//get # nodes (first line in file) and feed into circuit constructor
	getline(schematic, parseln);
	Circuit myCircuit(atoi(parseln.c_str()));

	//loop until no longer able to get input
		//each line of input is 1 wire
	getline(schematic, parseln);
	while (!schematic.fail()) {

		cout << "\nCreating wire: " << parseln;

		//get src node
```

```
            ele = parseln.substr(0, parseln.find("-"));
            src = atoi(ele.c_str());

            //get dest node
            ele = parseln.substr(parseln.find("-")+1, parseln.find("("));
            dest = atoi(ele.c_str());

            //get resistance between nodes
            ele = parseln.substr(parseln.find("(") + 1, parseln.find(")"));
            resistance = atoi(ele.c_str());

            //create specified wire
            myCircuit.wireJunctions(src, dest, resistance);

            //get next line
            getline(schematic, parseln);
        }

        //close input file
        schematic.close();

        //output resulting circuit data
        cout << "\n\nThe total resistance of the specified schematic is: "
                << myCircuit.calcResistance() << " Ohms.\n";

        system("pause");
        return 0;

}
```

Circuit.h

```
#ifndef CIRCUIT_H
#define		CIRCUIT_H

#include <vector>
#include "Junction.h"
#include "Wire.h"

using namespace std;

//~~~used to make recursion easier~~~
struct CollisionResult {
	double resistance;
	Junction* collisionNode;
};
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

class Circuit {

private:
	//list of junctions in the circuit
	vector<Junction*> junctions;

	//recursively calculate the resistance of the circuit
	double calcTotalResistance(Junction* relativeRoot);
	CollisionResult calcEqResistance(Junction* relativeRoot);

public:
	//***no direct getters and setters to maximize encapsulation***

	//constructor
	Circuit(int size);

	//set junction as null or battery
	void setGain(int junction, double voltage);

	//create a wire between two junctions
	void wireJunctions(int srcJunction, int destJunction, double resistance);

	//indirectly call the recursive calc resistance method
	double calcResistance();
};
#endif // !CIRCUIT_H
```

Circuit.cpp

```
#include "Circuit.h"

Circuit::Circuit(int size) {
	for (int i = 0; i < size; i++) {
		junctions.push_back(new Junction(i));
	}
}

void Circuit::setGain(int junction, double voltage) {
	if (junction < junctions.size()) {
		junctions.at(junction)->setVoltage(voltage);
	}
	else {
		//TODO: handle error
	}
}

void Circuit::wireJunctions(int srcJunction, int destJunction, double resistance) {
	//check valid wire
	if (srcJunction < junctions.size() && destJunction < junctions.size() && resistance > 0) {
		//create connection
		Wire* w = new Wire(junctions.at(srcJunction), junctions.at(destJunction),
resistance);
		junctions.at(srcJunction)->addConnection(w);
	}
	else {
		//TODO: handle error
	}
}

double Circuit::calcResistance() {
	return calcTotalResistance(junctions.at(0));
}

double Circuit::calcTotalResistance(Junction* relativeRoot) {

	vector<Wire*> forwardConnections = relativeRoot->getForwardConnections();
	int nForwardConnections = forwardConnections.size();

	//check # of possible traversals from current node and act accordingly
		//if only 1 --> add resistance and continue on
		//if > 1 --> calc parallel resistance and continue on
```

```
		//if < 1 --> handle error
	if (nForwardConnections < 1) {
		//TODO: throw error
	}
	else if (nForwardConnections == 1) {
		if (forwardConnections.at(0)->getDest()!=junctions.at(0)) {
			return forwardConnections.at(0)->getResistance() +
calcTotalResistance(forwardConnections.at(0)->getDest());
		}
		else {
			return forwardConnections.at(0)->getResistance();
		}
	}
	else {
		CollisionResult cr = calcEqResistance(relativeRoot);
		return cr.resistance + calcTotalResistance(cr.collisionNode);
	}
}

CollisionResult Circuit::calcEqResistance(Junction* relativeRoot) {

	vector<Wire*> forwardConnections = relativeRoot->getForwardConnections();
	int nForwardConnections = forwardConnections.size();

	//init values to calculate parallel resistance
	double tR = 0.0;
	double* resistance = new double[nForwardConnections];
	Junction** nodes = new Junction*[nForwardConnections];

	//init path resistances and node traversal points
	for (int i = 0; i < nForwardConnections; i++) {
		resistance[i] = forwardConnections.at(i)->getResistance();
		nodes[i] = forwardConnections.at(i)->getDest();
	}

	//detect the point where the parallel paths collide
	bool collided = false;
	while (!collided) {

		//find min of currently traversed nodes
		int minID = INT_MAX;
		int minIndex = -1;
		for (int i = 0; i < nForwardConnections; i++) {
```

```
                    if (nodes[i]->getID() < minID) {
                        minID = nodes[i]->getID();
                        minIndex = i;
                    }
                }

                //check # of possible traversals from smallest node and act accordingly
                    //if>1, find eqResistance from node to next collision
                    //if==1, add line resistance and move path down
                    //if<1, handle error
                if (nodes[minIndex]->getForwardConnections().size() > 1) {
                    CollisionResult cr = calcEqResistance(nodes[minIndex]);//get CR
                    resistance[minIndex] += cr.resistance;//adding eq resistance from this
node to it's next collision
                    nodes[minIndex] = cr.collisionNode;//set current path node to node of
collision
                }
                else if(nodes[minIndex]->getForwardConnections().size() == 1){
                    resistance[minIndex] +=
nodes[minIndex]->getForwardConnections().at(0)->getResistance();//adding next line resistance
                    nodes[minIndex] =
nodes[minIndex]->getForwardConnections().at(0)->getDest();//set current path node to next
node
                }
                else {
                    //TODO: handle error
                }

                //check if all nodes have now collided
                collided = true;
                for (int i = 1; i < nForwardConnections; i++) {
                    if (nodes[i] != nodes[0]) {
                        collided = false;
                        break;
                    }
                }

                //if all nodes have collided, calculate and return the resistance
                if (collided) {
                    for (int i = 0; i < nForwardConnections; i++) {
                        tR += 1.0 / resistance[i];
                    }
                    tR = 1 / tR;
```

```
                }

        }


        //generate collision result
        CollisionResult cr;
        cr.collisionNode = nodes[0];
        cr.resistance = tR;

        //delete dangling pointers before returning
        delete[] resistance;
        delete[] nodes;
        return cr;

}
```

Wire.cpp

```
#include "Wire.h"

Wire::Wire(Junction* src, Junction* dest, double resistance) {
        this->src = src;
        this->dest = dest;
        this->resistance = resistance;
}
```

Wire.h

```
#ifndef WIRE_H
#define	WIRE_H

class Junction;

class Wire {

	private:
		//trailing junction
		Junction* src;
		//leading junctions
		Junction* dest;
		//resistance of this wire
		double resistance;
		/*
		double capacitance;
		double inductance;
		*/

	public:
		//constructor
		Wire(Junction* src, Junction* dest, double resistance);
		//getters
		inline Junction* getSrc() { return src; }
		inline Junction* getDest() { return dest; }
		inline double getResistance() { return resistance; }
		/*
		inline double getCapacitance() { return capacitance; }
		inline double getInductance() { return inductance; }
		*/
		//setters
		inline void setSrc(Junction* src) { this->src = src; }
		inline void setDest(Junction* dest) { this->dest = dest; }
		inline void setResistance(double resistance) { this->resistance=resistance; }
		/*
		inline void setCapacitance() { this->capacitance=capacitance; }
		inline void setInductance() { this->inductance=inductance; }
		*/


};
#endif // !WIRE_H
```

Junction.h

```
#ifndef JUNCTION_H
#define JUNCTION_H

#include <vector>

using namespace std;

class Wire;

class Junction {

	private:
		//vector-list of wires moving away from this node
		vector<Wire*> connections;
		//voltage gain of this node
		double voltage;
		//id of node
		int id;

	public:
		//constructor
		Junction(int id, double voltage = 0.0);
		//getters
		inline bool isBattery() { return (voltage==0.0?true:false); }
		inline double getVoltage() { return voltage; }
		inline int getID() { return id; }
		inline vector<Wire*> getForwardConnections() { return connections; }
		//setters
		inline void setVoltage(double voltage) { this->voltage = voltage; }
		inline void setID(int id) { this->id = id; }
		inline void addConnection(Wire* w) { connections.push_back(w); }
};
#endif // !JUNCTION_H
```

Junction.cpp

```
#include "Junction.h"

Junction::Junction(int id, double voltage) {
	this->id = id;
	this->voltage = voltage;
}
```