University of Pittsburgh
Department of Electrical and Computer Engineering
ECE 0301 – Spring 2021

Assignment #12

Programming concepts:

- Aggregation
- Static member variables
- Returning objects from functions
- Pointers to objects
- Dynamic memory allocation
- Arrays of objects
- Inheritance: bases classes and derived classes

ECE concepts:

- Electric circuit simulation.
- The voltage divider and current divider circuits

Background: In Assignments 10 and 11, you developed a class for a time-domain signal, and wrote member functions to give objects from that class the functionality to define and visualize many kinds of analytic signals. For some applications, a single very large class with many powerful member functions is the right approach, but in other cases it is more appropriate to define multiple simple classes that interact in meaningful ways to model the behavior of complex systems.

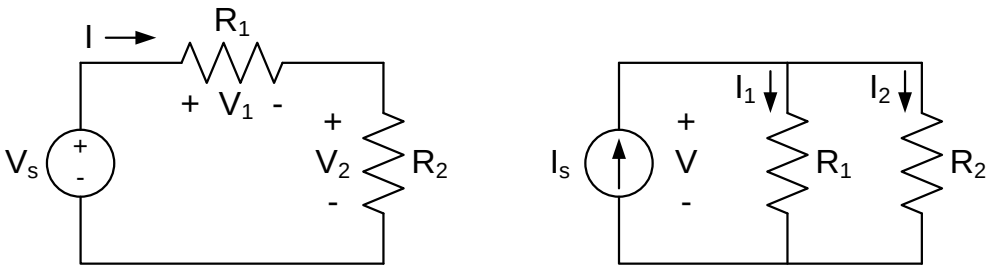

Figure 1: Voltage Divider and Current Divider circuits.

The purpose of this assignment is to illustrate some additional concepts related to classes and objects, and how you can take advantage of them to model systems of interest to Electrical and Computer Engineering, and other applications. In order to keep the length of the program reasonable, we will develop a system capable of simulating only the two simplest circuits you have studied, the voltage divider and the current divider, as shown in Figure 1. However, we will apply some programming concepts in a way that will allow for larger circuits to be simulated in the future, should we choose to do so.

An electrical network consists of an interconnection of components, and the points of connection are called nodes. If a network contains one or more closed loops, then it is an electric circuit, and we can use the familiar circuit analysis techniques to determine all of the voltages and currents. Therefore, in order to write an object-oriented program to simulate electric circuits, you will need three kinds of things, and you will write a class for each:

- Nodes. There must be a way to keep track of different nodes, and a way to know which nodes are associated with a given network. We will follow the convention used in node voltage analysis, by defining a ground node that by definition has a voltage of 0, and assign a voltage for all other nodes relative to ground.
- Components. Assume that all circuit components have two terminals, i.e. no MOSFETs, logic gates, integrated circuits, or anything with three or more terminals. All circuit components will be connected between two nodes, and it will be possible to compute the voltage across any component by taking the difference of the node voltages.
- Networks, which consists of collections of nodes and components. There is a "has-a" relationship here, as described in Gaddis, Section 14.7: a network has nodes, and a network has components. Therefore, the class for a network will use aggregation by containing instances of nodes and components.

Once you have written a class for a node, a class for a two-terminal component, and a class for a network that contains nodes and components, then you will be able to write a program that defines a network, sets the voltages at all of the nodes, and connects components between pairs of nodes. However, you will not be able to calculate the current flowing through the different components, because different types of components have different voltage-current relationships.

We only need to simulate voltage divider and current divider circuits, so we need only three kinds of components: independent voltages sources, independent current sources, and resistors. Each type of component has an "is-a" relationship, as described in Gaddis, Section 15.1:

- A resistor is a two-terminal component.
- An independent voltage source is a two-terminal component.
- An independent current source is a two-terminal component.

For this reason, we will have a base class for a two-terminal component, and derived classes for voltage source, a current source, and a resistor. Each component type will inherit the member variables from the component class, and will have additional member variables to model behavior specific to that type of component.

There are two types of networks to simulate, and each of these also has an "is-a" relationship:

- A voltage divider is a network.
- A current divider is a network.

Therefore, you will define a base class for a network, and derived classes for the voltage divider and current divider. There are also "has-a" relationships with the different types of components:

- A voltage divider has a voltage source and two resistors.
- A current divider has a current source and two resistors.

Therefore, the classes for the voltage divider and current divider will also use aggregation.

In designing classes for an application, it is important to define the kinds of information that must be stored in the member variables of each class, which other classes should be allowed to

access that information, and how they will do so. We will consider the eight classes required for this assignment, and how we will organize information for each of them, beginning with the simplest object: the node.

Nodes: What information do we need to store about a node?

- Each node has a voltage, so there must be a member variable to store the node voltage.
- A static member variable will be used to keep track of the total number of nodes that have been defined
- A unique integer index will be assigned to each node, by having an integer member variable for this purpose. This will allow us to provide information about that node to the user in a way that distinguishes it from information about other nodes.

Consult Gaddis Section 14.1 and Program 14-1 to see how to establish a static member variable that counts the number of objects that have been defined. The index member variable must also be set in the constructor, so that each instance of the class is assigned a unique value.

To what value should we initialize the node voltage in the constructor? Nodes do not come into existence with an inherent voltage, their voltages are determined by the circuit components that are connected between them. Therefore, it makes sense to assign all nodes a voltage of zero in the constructor, and allow the network to set the voltages of the nodes.

What other member functions will be needed for a node? There must be mutator and accessor functions for the node voltage, so that the network can set the voltage of any node, and nodes can report their voltages to networks and other objects. Ideally, we might want to restrict access to the mutator function so that only the network containing a node can set its voltage, but we will choose to make both functions public.

There must also be accessor functions for the number of nodes and the node index. We will not have mutator functions for the number of nodes and node index, so that these variables are only set by the constructor. Node number 2 will always be node number 2, and not even the network will be able to change that.

Finally, nodes must be able to report their data (index and voltage), so there will be a member function for this purpose. This function will write one line to a text file indicating the node index and node voltage.

Components: What information do we need to store about a component?

- There must be a way to distinguish between the terminals of a component. We will refer to the terminals as A and B.
- A component is connected between two nodes, so a component must store information about which node each terminal is connected to.
- We will use a static member variable to count the number of components, and a member variable to store a unique integer index for each component, just as we did with nodes.

What other functionality should a component have? A component should be able to report the voltage difference between its terminals, but there should *not* be a member variable to store this value, to avoid the possibility of "stale data" if the terminal voltage is not consistent with the node voltages. Therefore, a component must be able to reports its terminal voltage by taking the difference between the voltages at the nodes to which it is connected. We must have some convention with regard to the polarity of this voltage, so we will define the terminals voltage to be the voltage at terminal B minus the voltage at terminal A.

How will a component accomplish this task? The node class will have a public accessor member function that will return the node voltage, but the component must have access to the node in question in order to use the function. How can a component have access to the nodes to which it is connected? A component does not "have-a" node, so we should not make the nodes member variables of the component, but we can establish pointers to the nodes as member variables of the component.

To summarize, the component class will have four member variables:

- A static member variable to count the number of nodes,
- An integer variable to store a unique index for each component,
- A pointer to an object of the node class, which points to the node to which terminal A is connected.
- A pointer to an object of the node class, which points to the node to which terminal B is connected.

What functionality will objects from the component class need? What member functions will be required to achieve that functionality?

- The constructor will assign the component index, increment the component count, and set both node pointers to `nullptr`.
- There must be accessor functions for the component index and the number of components.
- There must be mutator functions for the node pointers. These will be used by the network to "connect" a component between two nodes.
- A component must be able to tell other objects which nodes it is connected to, so there must be a member function that returns the index of the node to which terminal A is connected, and another member function that returns the index of the node to which terminal B is connected.
- There must be a member function that returns the terminal voltage.
- Components must be able to report their information, so there must be a member function that writes a line of text to a file listing the component index and the indices of the nodes to which it is connected, and a second line of text listing the component index and terminal voltage.

Each component in an electric circuit will have some current flowing through it, and different types of components have different voltage-current relationships. It is not possible to compute the currents in a network of nodes and components without knowing which components are voltage sources, current sources and resistors. Therefore, we will not give the component class

any member functions to compute or reports currents, but we will reserve these capabilities for the derived classes.

Networks: What information do we need to store about a network? A network consists of nodes, and components connected between the nodes, so a network must be able to interact with nodes and components in ways that are meaningful. As we have already noted, a network must be able to set the voltage at each node in the network, through a mutator function of the node class for this purpose. To use that function, the network must have access to the nodes that it contains.

We will have a static member variable to count the number of networks, and a member variable to store a unique index for each network. As with nodes and components, having a unique index for each network will allow the program to report information about that network to the user.

Different types of networks contain different numbers of nodes, so the network class will have a member variable to store the number of nodes in the network. The network class will also have a dynamically-allocated array of objects from the node class, and a member variable that is a pointer to the beginning of this array. Therefore, a network will be able to set its node voltages by accessing the nodes in the array, and using the public member function of the node class to set each voltage.

A network must be able to connect components between nodes, by setting the node pointers for each component. Should there be a member variable that is an array of components in the network, as was done with nodes? Remember that all nodes are equivalent, whereas there will eventually be derived classes for three different types of components, and two different types of networks. The types of components used are different for the voltage divider and current divider networks, and for this reason we will *not* associate an array of components with each network.

What additional functionality will objects from the network class need? What member functions will be required to achieve that functionality?

- The constructor will assign the network index, increment the network count, and allocate memory for an array of three objects from the node class.
- A destructor will be needed, to release the memory allocated for the array of nodes when a network object is destroyed.
- There must be a member function that returns the network index.
- There must be a member function that returns the number of nodes in existence.
- There must be a member function that can return a pointer to any of the nodes that are contained in the network.
- There must be a member function to set the voltage at any of the nodes that are contained in the network.
- There must be a member function that returns the number of components in existence, and another member function to set the node pointers for a component to point to any of the nodes that are contained in the network. Each of these functions will take a pointer to a component as a parameter, so that the function can use that component to call the member functions of the component class.

- There must be a member function to report information about a network. This function writes three lines to a text file:
  - A line of text that reports the network index
  - A second line of text that indicates the number of networks in existence, and
  - A third line that indicates the number of nodes in existence.

  After writing these lines, this function calls the member function from the node class that reports node information, for each of the nodes that are contained in the network.

After developing the classes for a node, a component and a network, and defining all of the member functions described above, it will be possible to define networks like those shown in Figure 2. Network 0 contains three nodes, each labeled with a unique node index and the node voltage, and has three components connected in series, each labeled with a unique component index and the terminal voltage. Terminals A and B are also labeled for each component.

Network 1 contains two nodes, and three components connected in parallel, labeled similarly. The nodes and components in network 1 have different indices from those in network 0. Therefore, we can refer to nodes by their indices (which are unique for each node, even in different networks) or by their position within the array of nodes for a given network.

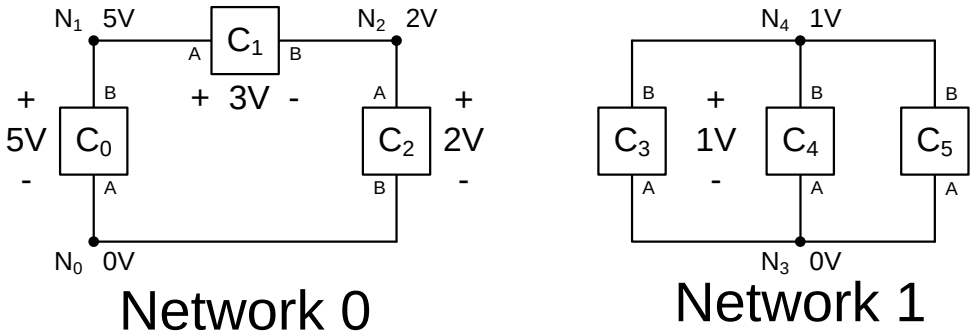

Figure 2: Two networks, with nodes and components

No currents are labeled in Figure 2, because the classes described so far do not have the capacity to determine what those currents should be. In order to complete the assignment, we will have to define derived classes for the different types of components and networks.

DC Voltage Sources: The DC voltage source will be a derived class, with the component as the base class. Therefore, the source will inherit the member variables of the base class (component count, component index, and node pointers) and will have access to the member functions that return the component count, component index, node indices and terminal voltage, as well as those that set the node pointers and write component information to a text file.

What additional information do we need to store about a DC voltage source that is not inherited from the base class?

- There must be a member variable to store the source voltage.
- When a DC voltage source is placed in a circuit, some amount of current will flow through it, so there must be a member variable to store the current.

What additional functionality will objects from the DC voltage source class need? What member functions will be required to achieve that functionality?

- The constructor must initialize the source voltage and the current. The source voltage can be set to any desired value, and a constructor will be developed that accepts a parameter for the source voltage. However, the current in the source cannot be known until it is connected to a circuit, so the constructor will initialize the current to zero.
- There must be mutator functions to set the source voltage and the current.
- There must be accessor functions that return the source voltage, the current flowing through the source, and the power delivered by the source.
- There must be a member function to report information about a DC voltage source. This function will call the member function to report information about a component, and then write three additional lines to a text file:
  - A line of text that gives the component index, and indicates that it is a DC voltage source,
  - A line of text that reports the current in the source and the direction of flow, and
  - A line of text that reports the power delivered by the source.

DC Current Sources: The DC current source will also be a derived class, with the component as the base class, and will inherit the member variables and functions of the base class.

What additional information do we need to store about a DC current source that is not inherited from the base class?

- There must be a member variable to store the source current.
- There will *not* be a member variable for the terminal voltage, because of the inherited member function for the base class that reports this voltage by taking the difference of the node voltages.

What additional functionality will objects from the DC current source class need? What member functions will be required to achieve that functionality?

- The constructor must initialize the source current, and a constructor will be developed that accepts a parameter for the source voltage.
- There must be a mutator function to set the source current.
- There must be accessor functions that return the source current, and the power delivered by the source.
- There must be a member function to report information about a DC current source. This function will call the member function to report information about a component, and then write three additional lines to a text file:
  - A line of text that gives the component index, and indicates that it is a DC current source,
  - A line of text that reports the voltage across the source and indicates which is the negative terminal, and
  - A line of text that reports the power supplied by the source.

Resistors: The resistor will also be a derived class, with the component as the base class, and will inherit the member variables and functions of the base class.

What additional information do we need to store about a resistor that is not inherited from the base class?

- There must be a member variable to store the resistance.
- There will *not* be a member variable for the terminal voltage, because of the inherited member function for the base class that reports this voltage by taking the difference of the node voltages.
- There will *not* be a member variable for the current, because this can be calculated from the terminal voltage and the resistance using Ohm's law.

What additional functionality will objects from the resistor class need? What member functions will be required to achieve that functionality?

- The constructor must initialize the resistance.
- There must be a mutator function to set the resistance.
- There must be accessor functions that return the resistance, the current flowing through the resistor, and the power dissipated in the resistor.
- There must be a member function to report information about a resistor. This function will call the member function to report information about a component, and then write three additional lines to a text file:
    - A line of text that gives the component index, and indicates that it is a resistor,
    - A line of text that reports the current in the resistor and the direction of flow, and
    - A line of text that reports the power dissipated in the resistor.

Voltage Dividers: The voltage divider will be a derived class, with the network as the base class. Therefore, the voltage divider will inherit the member variables of the base class (network count, network index, and the array of three nodes) and will have access to the member functions that return the node count, a pointer to one of the nodes in the network, and the component count, as well as those that set the node voltage, set the node pointers for a component, and write network information to a text file.

A voltage divider circuit, with labeled nodes and component terminals, is shown in Figure 3.

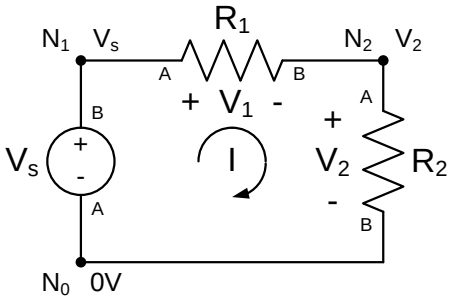

Figure 3: Voltage Divider

What additional information do we need to store about a voltage divider that is not inherited from the base class?

- Recall that the network class does not have any components. The member variables are an object from the DC voltage source class, and two objects from the resistor class.

What additional functionality will objects from the voltage divider class need? What member functions will be required to achieve that functionality?

- Any time the circuit conditions are changed, there must be a function to compute the voltages and currents in the circuit, and update the appropriate member variables of each object:
  - The voltage at node 0 must be set to 0 V, and the voltage at node 1 must be set to the DC source voltage.
  - The loop current must be computed, and the current for the DC source must be set to this value.
  - The voltage across resistor R_2 must be computed, and this value must be stored in the voltage member variable for node 2.
- The constructor must initialize the source voltage for the DC source, and the resistances of the two resistors. It must also set the node connections consistent with Figure 3:
  - The DC voltage source must have its terminal A node pointer point to node 0, and have its terminal B node pointer point to node 1.
  - Resistor R_1 must have its terminal A node pointer point to node 1, and have its terminal B node pointer point to node 2.
  - Resistor R_2 must have its terminal A node pointer point to node 2, and have its terminal B node pointer point to node 0.

  After making these connections, the constructor must call the function to compute the voltages and currents.
- There must be a member function that sets the DC source voltage to a desired value, and then calls the function to compute the voltages and currents.
- There must be a member function that sets the resistances to desired values, and then calls the function to compute the voltages and currents. This could be a single function for both resistors, or separate functions for R_1 and R_2.
- There must be a member function to report information about a voltage divider circuit. This function will call the member function to report information about a network, and then write two additional lines to a text file:
  - A line of text that reports the network index, and indicates that it this network is a voltage divider, and
  - A line of text that reports the number of components in existence.

  After writing these lines, this function calls the member function from the DC voltage source class that reports voltage source information, and then calls the member function from the resistor class that reports resistor information, for each of the two resistors in the circuit.

Current Dividers: The final class to be developed for this assignment is the current divider circuit. It will be a derived class, with the network as the base class, and will inherit the member variables and member functions of the base class.
A voltage divider circuit, with labeled nodes and component terminals, is shown in Figure 4.

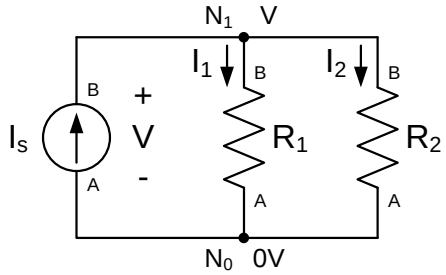

Figure 4: Current Divider

What additional information do we need to store about a current divider that is not inherited from the base class?

- The member variables are an object from the DC current source class, and two objects from the resistor class.

What additional functionality will objects from the voltage divider class need? What member functions will be required to achieve that functionality?

- Any time the circuit conditions are changed, there must be a function to compute the voltages and currents in the circuit, and update the appropriate member variables of each object:
  - The voltage at node 0 must be set to 0 V.
  - The voltage V in Figure 4 must be computed, by multiplying the source current by the equivalent resistance of R_1 and R_2 in parallel. This value must be stored in the voltage member variable for node 1.
  - Node 2 is not needed for the current divider circuit, so the voltage member variable for node 2 must be set to 0 V.
- The constructor must initialize the source current for the DC source, and the resistances of the two resistors. It must also set the node connections consistent with Figure 4:
  - All three components must have their terminal A node pointers point to node 0, and have their terminal B node pointers point to node 1.

  After making these connections, the constructor must call the function to compute the voltages and currents.
- There must be a member function that sets the DC source current to a desired value, and then calls the function to compute the voltages and currents.
- There must be a member function that sets the resistances to desired values, and then calls the function to compute the voltages and currents. This could be a single function for both resistors, or separate functions for R_1 and R_2.
- There must be a member function to report information about a current divider circuit. This function will call the member function to report information about a network, and then write two additional lines to a text file:
  - A line of text that reports the network index, and indicates that it this network is a current divider, and

- A line of text that reports the number of components in existence.

After writing these lines, this function calls the member function from the DC current source class that reports current source information, and then calls the member function from the resistor class that reports resistor information, for each of the two resistors in the circuit.

Instructions: Develop software according to the following specifications and submit whatever portion you have completed to the class repository before midnight on the due date.

1. Create a class specification file and a class implementation file for a node class named `Node`. To begin, implement only two of the private member variables:
   - A static member variable of type `int` to store the number of nodes in existence, and
   - A member variable of type `int` to store the node index.

   At this step, implement only the member functions needed to keep track of the number of nodes, and assign a unique index to each node.
   - The static member variable that stores the number of nodes must be initialized to zero outside the class.
   - The constructor must set the node index and increment the number of nodes in existence.
   - There must be an accessor functions that return the number of nodes and the node index. Like all accessor functions, the prototype and header must include the key word `const` to indicate that they will not modify any member variables.

   Write a program file containing a `main` function that tests the node class, by performing the following steps:
   - Write the following introductory message to standard output
     ```
     ECE 0301 - Electrical Network Simulation
     ```
   - Define three nodes, and for each of them:
     - Print a message to standard output to report the index of the node that was just created.
     - Print a message to standard output that reports the number of nodes in existence.

   Test your program, making sure that the nodes are assigned indices 0, 1, 2, etc., and that the number of nodes is counted accurately.

2. At this step, we will give nodes the ability to store and report their node voltage.
   - Add a member variable to the node class of type `double` to store the node voltage.
   - Write a mutator function to set the node voltage to a desired value.
   - Add a statement to the constructor to initialize the node voltage to 0.
   - Write an accessor function to return the node voltage. Make sure to include the key word `const` in the function prototype and header.

- Write a member function to report information about a node. This function takes a parameter that is an `ofstream` object, passed by reference, which has already been used to open a file for writing. The function writes one line of text to the file that reports the node index and node voltage, with the following format:
  ```
  Voltage at node 2 = 2.5
  ```
  Make sure to include the key word `const` in the prototype and header for this function, because it does not modify any member variables.

Modify your `main` function so that it performs the following actions:

- Define an `ofstream` object, and use it to open a file named
  ```
  ECE 0301 - Electrical Network Reports.txt
  ```
- Write the introductory message to the file
  ```
  ECE 0301 - Electrical Network Simulation
  ```
- Define a node
- Define a second node, and set its voltage to 5 V.
- Define a third node, and set its voltage to 2.5 V.
- Write a line of text to the file that reports the number of nodes in existence,
- For each of the three nodes, call the member function from the node class that prints node information to a text file.
- Close the text file.

Test your program and examine the output file. The correct output is:

```
ECE 0301 - Electrical Network Simulation
There are currently 3 nodes in existence.
Voltage at node 0 = 0.
Voltage at node 1 = 5.
Voltage at node 2 = 2.5.
```

3. Create a class specification file and a class implementation file for a network class. The private member variables for the class are:
   - A static member variable of type `int` to store the number of networks in existence,
   - A member variable of type `int` to store the network index,
   - A member variable of type `int` to store the number of nodes in this network, and
   - A pointer to the beginning of an array of nodes.

   At this step, we will give the network class only the member functions needed to interact with the nodes it contains, and report information to the user.
   - The constructor must accept a parameter of type `int` representing the number of nodes in the network. The default value for this parameter must be 2. The actions to be performed by the constructor are:
     - Assign the network index,
     - Increment the number of networks in existence,
     - Set the number of nodes in the network to value of the parameter, and
     - Allocate memory for an array of nodes, with the number of elements equal to the value of the member variable for the number of nodes in this network.

- There must be a destructor to release the memory allocated for the array of nodes.
- There must be a `const` member function that returns the network index.
- There must be a `const` member function that returns the number of nodes in this network.
- There must be a `const` member function that returns the number of nodes in existence. This can be accomplished by accessing any of the nodes in the network, and calling the member function from the node class that returns the number of nodes in existence.
- There must be a member function to set the voltage at any of the nodes in the network. This function takes a parameter of type `int` that indicates which element in the array of nodes is to have its voltage altered, and a parameter of type `double` containing the desired voltage.
- There must be a member function to report information about a network. This function takes a parameter that is an `ofstream` object, passed by reference, which has already been used to open a file for writing. This function writes six lines to the text file:
  - An empty line
  - A line of 48 dashes
  - A line of text that reports the network index
  - A line of text that reports the number of networks in existence,
  - A line of text that reports the number of networks in this network, and
  - A line that indicates the number of nodes in existence.

  After writing these lines, this function calls the member function from the node class that reports node information, for each of the nodes that are contained in the network. The format for the output file is shown below.

Modify your `main` function so that it performs the following actions:

- Define an `ofstream` object, use it to open the text file, and write the introductory message to the text file.
- Define a network with 3 nodes, and set it to have node voltages of 0.0 V, 5.0 V, and 2.5 V.
- Call the member function of the network class to write information about the network to the text file.
- Define a second network with 2 nodes, and set it to have node voltages of 0.0 V and 3.3 V.
- Call the member function of the network class to write information about the second network to the text file.
- Close the text file.

Test your program to ensure that the text file contains correct information about the two networks and all six of the nodes. Do all networks and nodes have a unique index? Are all voltages correct? The format for the output file at this step is shown in the next page.

```
ECE 0301 - Electrical Network Simulation

------------------------------------------------

Data for Electric Network # 0:
At present, there are 1 Networks in existence.
At present, there are 3 nodes in existence.
Network # 0 contains 3 nodes.
Voltage at node 0 = 0.
Voltage at node 1 = 5.
Voltage at node 2 = 2.5.

------------------------------------------------

Data for Electric Network # 1:
At present, there are 2 Networks in existence.
At present, there are 5 nodes in existence.
Network # 1 contains 2 nodes.
Voltage at node 3 = 0.
Voltage at node 4 = 3.3.
```

4. Create a class specification file and a class implementation file for a two-terminal component class. The private member variables for the class are:
    - A static member variable of type `int` to store the number of components in existence,
    - A member variable of type `int` to store the component index,
    - A pointer to an object from the node class, which will point to the node to which terminal A is connected, and
    - A pointer to an object from the node class, which will point to the node to which terminal B is connected.

    At this step, we will give the component class only the member functions needed to interact with nodes.
    - The constructor must assign the component index, increment the number of components in existence, and initialize both node pointers to `nullptr`.
    - There must be `const` accessor functions for the component index and the number of components in existence.
    - There must be mutator functions for the node pointers. Each of these functions takes a pointer to an object from the node class as a parameter, and sets the node pointer member variable to the value of the parameter.
    - There must be a `const` member function that returns the index of the node to which terminal A is connected, and another `const` member function that returns the index of the node to which terminal B is connected.

Modify your `main` function so that it performs the following actions:

- Define an `ofstream` object, use it to open the text file, and write the introductory message to the text file.
- Define two nodes.
- Define a component.
- Call the member function of the component class that sets the node pointer for terminal A, so that this pointer points to the first node.
- Call the member function of the component class that sets the node pointer for terminal B, so that this pointer points to the second node.
- Call the member function of the node class to write information about the first node to the text file.
- Call the member function of the node class to write information about the second node to the text file.
- Write a line of text to the file to report the number of components in existence.
- Write another line of text to the file to report component index, and the indices of the nodes to which the component terminals are connected.
- Close the text file.

Test your program, and confirm that the text file contains the following information.

```
ECE 0301 - Electrical Network Simulation
Voltage at node 0 = 0.
Voltage at node 1 = 0.
At present, there are 1 components in existence.
Component # 0 is connected between node 0 and node 1.
```

5. In this step, we will allow networks to connect a component between two nodes contained in the network.

   Add a member function to the network class that takes three parameters (a pointer to an object from the component class and two parameters of type `int`) and returns nothing. This function sets the node pointers for the component referenced by the first parameter. The node pointer for terminal A is set to the element of the node array indexed by the first `int` parameter, and the node pointer for terminal B is set to the element of the node array indexed by the second `int` parameter.

   Add a member function to the component class that writes information about a component to a text file. This function takes a parameter that is an `ofstream` object, passed by reference, which has already been used to open a file for writing. This function writes one line to the text file to report the component index, and the indices of the nodes to which the component terminals are connected.

   Modify your `main` function so that it defines the network shown in Figure 5, and writes information about it to a text file, by performing the following actions:

   - Define an `ofstream` object, use it to open the text file, and write the introductory message to the text file.

- Define a network with 3 nodes.
- Define 3 components.
- Call the member function from the network class that sets the node pointers for a component, so that C_0 is connected between node 0 and node 1, as shown in Figure 5. Repeat for the other two components.
- Call the member function that writes information about the network to the output file.
- Use component C_0 to call the member function that writes information about a component to the output file. Repeat for the other two components.
- Close the output file.

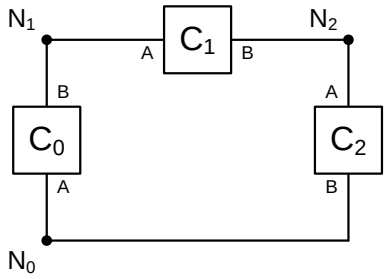

Figure 5: Network with 3 nodes and 3 components, connected in series.

Test your program to ensure that the text file contains the following information:

```
ECE 0301 - Electrical Network Simulation

------------------------------------------------

Data for Electric Network # 0:
At present, there are 1 Networks in existence.
At present, there are 3 nodes in existence.
Network # 0 contains 3 nodes.
Voltage at node 0 = 0.
Voltage at node 1 = 0.
Voltage at node 2 = 0.
Component # 0 is connected between node 0 and node 1.
Component # 1 is connected between node 1 and node 2.
Component # 2 is connected between node 2 and node 0.
```

Note: Before proceeding, recall that there is a "has a" relationship between a network and its nodes and components:

- A network has nodes. For example, the network in Figure 5 has 3 nodes.
- A network has components. For example the network in Figure 5 has 3 components.

We have made this relationship formal for the nodes in a network, by having a dynamically-allocated array of objects from the node class as one of the member variables for the network class. However, we have not chosen to do this for the component class, so our network class does not "have" any components, because no components are created when a network is created, and the network has no way to access the components. Instead, the "has a" relationship will be made formal with the derived classes for the voltage divider and current

divider networks, and the derived classes for the voltage source, current source and resistor components.

6. In this step, we will give the component class the ability to compute, return and report its terminal voltage. This will be done by using the node pointer for each terminal to reference the nodes that are connected to them.

   Write a member function for the component class that takes no parameters, and returns the terminal voltage, as follows:

   - Use the node referenced by the node pointer for terminal A to call the member function from the node class that returns the node voltage. This gives the voltage at terminal A.
   - Repeat to obtain the voltage at terminal B.
   - Return the voltage at terminal B minus the voltage at terminal A.

   Modify the member function of the component class that writes component information to a text file, so that it performs the following steps:

   - Write a line of text to the file to report the component index, and the indices of the nodes to which the component terminals are connected.
   - Write two lines of text to report the voltage across the component, and which node is the negative terminal. For example, in Figure 6, the voltage across C_0 is 5V, with the negative terminal at node 1. The terminal voltage returned by the member function described above may be negative, but **this function must always report a positive value** for the voltage, and report the polarity by identifying the negative terminal. See below for the proper text format.

   Modify your `main` function so that, after defining the network and connecting the components in series, the voltage at node 1 is set to -5 V, and the voltage at node 2 is set to +1.25 V.

   If your program is correct, it will create the network shown in Figure 6, and write information about it to the output file.

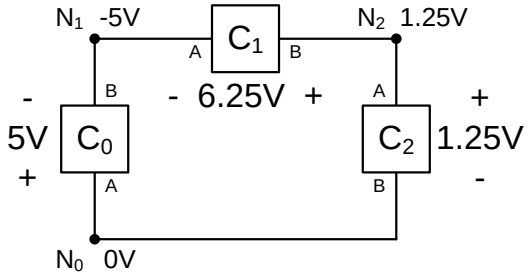

Figure 6: Series network with 3 nodes and 3 components, each reporting its voltage.

The correct output file is as follows:

```
ECE 0301 - Electrical Network Simulation

-----------------------------------------------

Data for Electric Network # 0:
At present, there are 1 Networks in existence.
At present, there are 3 nodes in existence.
Network # 0 contains 3 nodes.
Voltage at node 0 = 0.
Voltage at node 1 = -5.
Voltage at node 2 = 1.25.
Component # 0 is connected between node 0 and node 1.
The Voltage across Component # 0 = 5 Volts,
with the negative terminal at node 1.
Component # 1 is connected between node 1 and node 2.
The Voltage across Component # 1 = 6.25 Volts,
with the negative terminal at node 1.
Component # 2 is connected between node 2 and node 0.
The Voltage across Component # 2 = 1.25 Volts,
with the negative terminal at node 0.
```

When you are certain your program is correct, save you main function in a file named

```
ece0301_inclass12_step06.cpp
```

Submit this file, and all of the following files, to the class repository.

```
Node.h
Node.cpp
Component.h
Component.cpp
Network.h
Network.cpp
```

7. Create a class specification file and a class implementation file for a DC voltage source class named `DCVoltageSource`. This class must be a derived class, with the component class as the base class. At this step, we will only give a DC source the capability to set, return and report the source voltage. The only private member variable of the class is of type `double`, to store the source voltage.

   Write four member functions for the class:

   - The constructor must initialize the source voltage to zero.
   - There must be a mutator function to set the source voltage, and a `const` accessor function to return the source voltage.
   - There must be a member function to write information about a DC voltage source to a text file. This function takes a parameter that is an `ofstream` object, passed by reference, which has already been used to open a file for writing. This function writes one line to the output file that reports the component index, tells the user that it

is a DC voltage source, and reports the source voltage. Next, this function must call the member function for the component class that writes component information to a text file. See below for the proper text format.

Modify your `main` function so that it defines the network shown in Figure 7, and writes information about it to a text file, by performing the following actions:

- Define an `ofstream` object, use it to open the text file, and write the introductory message to the text file.
- Define a network with 3 nodes.
- Define a DC voltage source, and set the source voltage to $V_s = 12$ V.
- Call the member function from the network class that sets the node pointers for a component, so that the DC voltage source is connected between node 0 and node 1, as shown in Figure 7.
- Define two more components.
- Call the member function from the network class that sets the node pointers for a component, so that C_1 is connected between node 1 and node 2, as shown in Figure 7. Repeat for C_2.
- Set the voltage at node 1 to V_s and the voltage at node 2 to $V_s / 3$.
- Call the member function that writes information about the network to the output file.
- Use the DC voltage source to call the member function that writes information about a DC voltage source to the output file.
- Use C_1 to call the member function that writes information about a component to the output file. Repeat for C_2.
- Close the output file.

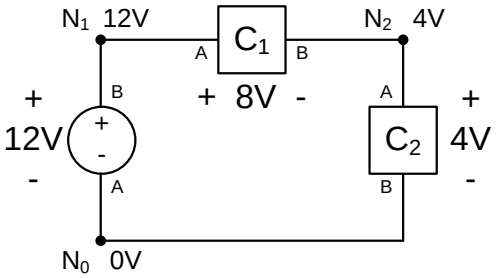

Figure 7: Series network with a DC Voltage source.

The correct output file is as follows:

```
ECE 0301 - Electrical Network Simulation

------------------------------------------------

Data for Electric Network # 0:
At present, there are 1 Networks in existence.
At present, there are 3 nodes in existence.
Network # 0 contains 3 nodes.
Voltage at node 0 = 0.
```

```
Voltage at node 1 = 12.
Voltage at node 2 = 4.

Component # 0 is a DC Voltage Source, Vs = 12 Volts.
Component # 0 is connected between node 0 and node 1.
The Voltage across Component # 0 = 12 Volts,
with the negative terminal at node 0.
Component # 1 is connected between node 1 and node 2.
The Voltage across Component # 1 = 8 Volts,
with the negative terminal at node 2.
Component # 2 is connected between node 2 and node 0.
The Voltage across Component # 2 = 4 Volts,
with the negative terminal at node 0.
```

8. Create a class specification file and a class implementation file for a resistor class. This class must be a derived class, with the component class as the base class. At this step, we will only give a resistor the capability to set, return and report the resistance. The only private member variable of the class is of type `double`, to store the resistance.

   Write four member functions for the class:
   - The constructor must initialize the resistance to 1 kΩ.
   - There must be a mutator function to set the resistance, and a `const` accessor function to return the resistance.
   - There must be a member function to write information about a resistor to a text file. This function takes a parameter that is an `ofstream` object, passed by reference, which has already been used to open a file for writing. This function writes one line to the output file that reports the component index, tells the user that it is a resistor, and reports the resistance. Next, this function must call the member function for the component class that writes component information to a text file. See below for the proper text format.

   Modify your `main` function so that it defines the network shown in Figure 8, and writes information about it to a text file, by performing the following actions:
   - Define an `ofstream` object, use it to open the text file, and write the introductory message to the text file.
   - Define a network with 3 nodes.
   - Define a DC voltage source, and set the source voltage to $V_s = 12$ V.
   - Call the member function from the network class that sets the node pointers for a component, so that the DC voltage source is connected between node 0 and node 1, as shown in Figure 8.
   - Define a resistor, and set the resistance to 200 Ω.
   - Call the member function from the network class that sets the node pointers for a component, so that the 200 Ω resistor is connected between node 1 and node 2, as shown in Figure 8.
   - Define a resistor, and set the resistance to 100 Ω.

- Call the member function from the network class that sets the node pointers for a component, so that the 100 Ω resistor is connected between node 2 and node 0, as shown in Figure 8.
- Set the voltage at node 1 to V_s and the voltage at node 2 to $V_s / 3$. (Note that this is consistent with the resistor values.)
- Call the member function that writes information about the network to the output file.
- Use the DC voltage source to call the member function that writes information about a DC voltage source to the output file.
- Use the 200 Ω resistor to call the member function that writes information about a resistor to the output file. Repeat for the 100 Ω resistor.
- Close the output file.

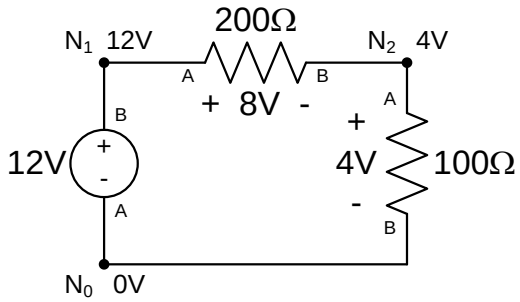

Figure 8: Voltage divider network with a DC Voltage source and two series resistors.

The correct output file is as follows:

```
ECE 0301 - Electrical Network Simulation

------------------------------------------------

Data for Electric Network # 0:
At present, there are 1 Networks in existence.
At present, there are 3 nodes in existence.
Network # 0 contains 3 nodes.
Voltage at node 0 = 0.
Voltage at node 1 = 12.
Voltage at node 2 = 4.

Component # 0 is a DC Voltage Source, Vs = 12 Volts.
Component # 0 is connected between node 0 and node 1.
The Voltage across Component # 0 = 12 Volts,
with the negative terminal at node 0.

Component # 1 is a Resistor, R = 200 Ohms.
Component # 1 is connected between node 1 and node 2.
The Voltage across Component # 1 = 8 Volts,
with the negative terminal at node 2.

Component # 2 is a Resistor, R = 100 Ohms.
```

```
Component # 2 is connected between node 2 and node 0.
The Voltage across Component # 2 = 4 Volts,
with the negative terminal at node 0.
```

9. In this step, we will modify the classes for a DC voltage source and a resistor, so that they can calculate and report their currents and powers. This will allow us to write a main function that performs a complete simulation of a simple circuit.

   Make the following modifications to the DC voltage source class:
   - Add a member variable of type `double` to store the current drawn from the source.
   - Add a mutator function to set the current.
   - Add a `const` accessor function that returns the current.
   - Add a `const` member function that returns the power supplied by the source.
   - Modify the member function that writes information about a DC voltage source to the text file so that it performs the following actions:
     - Report the component index, indicate that this component is a DC voltage source, and report the source voltage.
     - Call the member function from the component class to write information about a component to the text file.
     - Report the current and direction of flow, by indicating which node the current is flowing from, and which node the current is flowing to. For example, in Figure 8, the current drawn from the voltage source is 0.04 A, and this current flows from node 0 to node 1. The value stored in the current member variable may be negative, but **this function must always report a positive value** for the current, and report the direction of flow correctly.
     - Report the power supplied by the source.

   Make the following modifications to the resistor class:
   - Add a `const` member function that returns the current flowing through the resistor. This function calls the member function from the component class to return the terminal voltage, computes the current through Ohm's law, and returns the current.
   - Add a `const` member function that returns the power dissipated in the resistor.
   - Modify the member function that writes information about a resistor to the text file so that it performs the following actions:
     - Report the component index, indicate that this component is a resistor, and report the resistance.
     - Call the member function from the component class to write information about a component to the text file.
     - Report the current and direction of flow, by indicating which node the current is flowing from, and which node the current is flowing to. The current returned by the member function may be negative, but **this function must always report a positive value** for the current, and report the direction of flow correctly.
     - Report the power dissipated in the resistor.

   The only change to your `main` function at this step will be to compute the current drawn from the voltage source, and call the member function from the DC voltage source class to

set the current to this value. These statements should be placed after the statements that set the node voltages, and before the statements that call member functions to write information to the output file.

The correct output file is as follows:

```
ECE 0301 - Electrical Network Simulation

-----------------------------------------------

Data for Electric Network # 0:
At present, there are 1 Networks in existence.
At present, there are 3 nodes in existence.
Network # 0 contains 3 nodes.
Voltage at node 0 = 0.
Voltage at node 1 = 12.
Voltage at node 2 = 4.

Component # 0 is a DC Voltage Source, Vs = 12 Volts.
Component # 0 is connected between node 0 and node 1.
The Voltage across Component # 0 = 12 Volts,
with the negative terminal at node 0.
The current in this DC Voltage Source = 0.04 Amps,
flowing from Node 0 to Node 1.
The power supplied by this DC Voltage Source = 0.48 Watts.

Component # 1 is a Resistor, R = 200 Ohms.
Component # 1 is connected between node 1 and node 2.
The Voltage across Component # 1 = 8 Volts,
with the negative terminal at node 2.
The current in this Resistor = 0.04 Amps,
flowing from Node 1 to Node 2.
The power dissipated in this Resistor = 0.32 Watts.

Component # 2 is a Resistor, R = 100 Ohms.
Component # 2 is connected between node 2 and node 0.
The Voltage across Component # 2 = 4 Volts,
with the negative terminal at node 0.
The current in this Resistor = 0.04 Amps,
flowing from Node 2 to Node 0.
The power dissipated in this Resistor = 0.16 Watts.
```

When you are certain your program is correct, save you main function in a file named

```
ece0301_inclass12_step09.cpp
```

Submit this file, and all of the following files, to the class repository.

```
Node.h
Node.cpp
Component.h
```

```
Component.cpp
Network.h
Network.cpp
DCVoltageSource.h
DCVoltageSource.cpp
Resistor.h
Resistor.cpp
```

10. Create a class specification file and a class implementation file for a voltage divider class. This class must be a derived class, with the network class as the base class. Each voltage divider circuit has three nodes. The private member variables of the class are an object from the DC voltage source class, and two objects from the resistor class.

    At this step, we will give a voltage divider the capability to:
    - Set the node pointers for the voltage source and resistors,
    - Set the source voltage for the voltage source,
    - Set the resistance for the resistors, and
    - Report information about the circuit, its nodes, and its components.

    Add a member function to the network class that returns the pointer to one of the nodes contained in the network. This function accepts a variable of type `int` as the only parameter, and returns a pointer to an object from the node class. This function returns a pointer to the element from the node array member variable that is indexed by the value of the parameter. This is function is needed by the constructors of the voltage divider class to set the node pointers for components.

    Write five member functions for the voltage divider class:
    - The default constructor for this class must:
      - Call the constructor for the network class, with 3 nodes,
      - Set the source voltage of the voltage source to 1 V,
      - Set the resistance of both resistors to 1 kΩ,
      - Connect the DC voltage course between nodes 0 and 1,
      - Connect R_1 between nodes 1 and 2, and
      - Connect R_2 between nodes 2 and 0.
    - There must also be a constructor that accepts three arguments of type `double` that contain desired values for the source voltage and resistances, respectively. This constructor is identical to the default constructor, except that it sets the source voltage and resistances to the values of the parameters.
    - There must be a mutator function for the source voltage.
    - There must be a mutator function that sets the resistances of R_1 and R_2.
    - There must be a member function to write information about a voltage divider to a text file. This function takes a parameter that is an `ofstream` object, passed by reference, which has already been used to open a file for writing.

      Examine the member function from the network class that writes information to the output file. In previous steps, the first actions performed by this function were to

write a blank line to the file, followed by a line of 48 dashes, followed by another blank line, followed by the line

```
Data for Electric Network # 0:
```

Cut these statements from the function for the network class, and paste them at the beginning of the corresponding function for the voltage divider class. Then add statements to this function that:

- Write a line to the file reporting the network index, and indicating that the network is a voltage divider,
- Call the member function from the network class to write network information to the file,
- Write a line to the file to report the number of components in existence, by using any of the components in the circuit to call the member function from the component class that returns the number of components,
- Call the member function from the DC voltage source class that writes information to a text file,
- Use R_1 to call the member function from the resistor class that writes resistor information to a text file, and
- Use R_2 to call the member function from the resistor class that writes resistor information to a text file.

Modify your `main` function so that it defines the network shown in Figure 8, and writes information about it to a text file, by performing these actions:

- Define an `ofstream` object, use it to open the text file, and write the introductory message to the text file.
- Define a voltage divider object with the source voltage and resistances shown in Figure 8.
- Call the member function from the voltage divider class that writes information to the file.
- Close the text file.

Please note that the information contained in the file at this step will be different from what you obtained in step 8. The nodes should all report a voltage of zero, because the voltage divider does not yet have the capability to set node voltages, which also means that the components should report that their voltage, current and power are all zero.

11. In this step, we will complete the development of the voltage divider class, by giving it the capability to determine all of the voltages and currents in the circuit, and write additional information to the text file.

    Modifications to the voltage divider class:

    - Write a new member function that takes no arguments and returns nothing. This function calculates all of the voltages and currents in the circuit, and saves the values in the appropriate member variables of the nodes and components, by performing the following actions:
      - Set the voltage at node 0 to 0 V.
      - Set the voltage at node 1 equal to the source voltage of the DC voltage source.

  - Compute the loop current for the voltage divider circuit.
  - Set the current member variable for the DC voltage source equal to the value of the loop current.
  - Set the voltage at node 2 equal to the loop current times the resistance of R_2.
- The function to compute voltages and currents and set member variables must be called as the last step of every member function that changes the circuit conditions:
  - Both constructors,
  - The mutator function that sets the source voltage, and
  - The mutator function that sets the resistances.

You should not have to modify your `main` function at all for this step. The text file generated by your program should contain all of the information shown in Figure 8, with all voltages and polarities reported correctly. The correct output file is as follows:

```
ECE 0301 - Electrical Network Simulation

------------------------------------------------

Data for Electric Network # 0:
Network # 0 is a Voltage Divider.
At present, there are 1 Networks in existence.
At present, there are 3 nodes in existence.
Network # 0 contains 3 nodes.
Voltage at node 0 = 0.
Voltage at node 1 = 12.
Voltage at node 2 = 4.
At present, there are 3 components in existence.

Component # 0 is a DC Voltage Source, Vs = 12 Volts.
Component # 0 is connected between node 0 and node 1.
The Voltage across Component # 0 = 12 Volts,
with the negative terminal at node 0.
The current in this DC Voltage Source = 0.04 Amps,
flowing from Node 0 to Node 1.
The power supplied by this DC Voltage Source = 0.48 Watts.

Component # 1 is a Resistor, R = 200 Ohms.
Component # 1 is connected between node 1 and node 2.
The Voltage across Component # 1 = 8 Volts,
with the negative terminal at node 2.
The current in this Resistor = 0.04 Amps,
flowing from Node 1 to Node 2.
The power dissipated in this Resistor = 0.32 Watts.

Component # 2 is a Resistor, R = 100 Ohms.
Component # 2 is connected between node 2 and node 0.
The Voltage across Component # 2 = 4 Volts,
with the negative terminal at node 0.
```

```
The current in this Resistor = 0.04 Amps,
flowing from Node 2 to Node 0.
The power dissipated in this Resistor = 0.16 Watts.
```

12. In this step, you will write the last two classes for this assignment, the DC current source and the current divider. These classes will be constructed in ways that are directly analogous to the DC voltage course and voltage divider classes. Create a class specification file and a class implementation file for each.

    The DC current source class is a derived class, with the component class as the base class, and it has only one member variable of type `double` to store the source current. The public member functions for the DC current source class are as follows:

    - The constructor initializes the source current to zero.
    - There must be a mutator function to set the source current.
    - There must be a `const` accessor function that returns the source current.
    - There must be a `const` member function that returns the power supplied by the source.
    - There must be a member function to write information about a DC current source to a text file. This function takes a parameter that is an `ofstream` object, passed by reference, which has already been used to open a file for writing, and performs the following steps
        - Report the component index, indicate that this component is a DC current source, and report the source current.
        - Call the member function from the component class to write information about a component to the text file.
        - Report the current and direction of flow, by indicating which node the current is flowing from, and which node the current is flowing to. The value stored in the current member variable may be negative, but **this function must always report a positive value** for the current, and report the direction of flow correctly.
        - Report the power supplied by the source.

    The current divider class is a derived class, with the network class as the base class. Each current divider circuit has two nodes. The private member variables of the class are an object from the DC current source class, and two objects from the resistor class. The public member functions for the current divider class are as follows.

    - The default constructor for this class must:
        - Call the constructor for the network class, with 2 nodes,
        - Set the source current of the current source to 1 A,
        - Set the resistance of both resistors to 1 kΩ, and
        - Connect all three components between nodes 0 and 1.
    - There must also be a constructor that accepts three arguments of type `double` that contain desired values for the source current and resistances, respectively. This constructor is identical to the default constructor, except that it sets the source voltage and resistances to the values of the parameters.
    - There must be a mutator function for the source current.
    - There must be mutator function(s) for the resistances of R_1 and R_2.

- There must be a member function that calculates all of the voltages and currents in the circuit, and writes the values to the appropriate member variables of the nodes and components, by performing the following steps:
    - Set the voltage at node 0 to 0 V.
    - Compute the equivalent resistance of the two parallel resistors.
    - Set the voltage at node 1 equal to the source current of the DC current source, multiplied by the equivalent resistance.
- There must be a member function to write information about a current divider to a text file. This function takes a parameter that is an `ofstream` object, passed by reference, which has already been used to open a file for writing. This function:
    - Writes a blank line to the file, followed by a line of 48 dashes, followed by another blank line,
    - Writes a line to report the network index,
    - Writes another line to report the network index, and that this network is a current divider,
    - Calls the member function for the network class that writes information about a network to a text file,
    - Writes a line to the file to report the number of components in existence, by using any of the components in the circuit to call the member function from the component class that returns the number of components,
    - Calls the member function from the DC current source class that writes information to a text file,
    - Uses R_1 to call the member function from the resistor class that writes resistor information to a text file, and
    - Uses R_2 to call the member function from the resistor class that writes resistor information to a text file.

Modify your `main` function so that it defines the network shown in Figure 9, and writes information about it to a text file, by performing these actions:

- Define an `ofstream` object, use it to open the text file, and write the introductory message to the text file.
- Define a current divider object with the source voltage and resistances shown in Figure 9.
- Call the member function from the voltage divider class that writes information to the file.
- Close the text file.

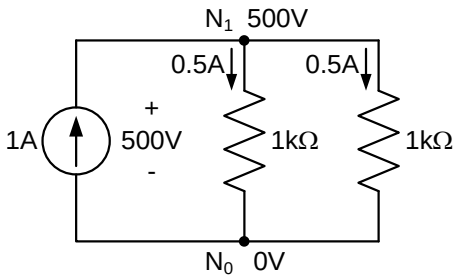

Figure 9: Current divider network with a DC Voltage source and two parallel resistors.

13. In the last step, you will give your program the capability to read an input file describing circuits to be simulated, and then simulate them and write the results to the output file.

    Create a new text file named "`ECE 0301 - Circuits to Simulate.txt`" and type the following four lines into it:

    ```
    Voltage Divider
    12
    200
    100
    ```

    Modify your main function so that it performs the following actions:

    - Define an i`fstream` object, and use it to open the new text file you just created.
    - Define an `ofstream` object, use it to open the text file, and write the introductory message to the text file.
    - Iterate through this loop until there are no more lies to read from the input file:
      - Read the next line from the input file.
      - If the line just read is not "`Voltage Divider`" or "`Current Divider`" print the following message to standard output, and exit the program with a failure code:

        ```
        ERROR! Invalid network type.
        ```

      - If the line just read from the file is "`Voltage Divider`":
        - Read the next three lines, and store the values as V_s, R_1 and R_2.
        - Define a voltage divider object, and pass the values for V_s, R_1 and R_2 to the constructor.
        - Call the member function from the voltage divider class that writes information to the output file.
      - If the line just read from the file is "`Current Divider`":
        - Read the next three lines, and store the values as I_s, R_1 and R_2.
        - Define a voltage divider object, and pass the values for I_s, R_1 and R_2 to the constructor.
        - Call the member function from the voltage divider class that writes information to the output file.
    - Close the input and output files.

    Simulate the circuits shown in Figure 10, and confirm that your program produces the correct output in each case. Power values are not shown, but should be confirmed. An input file that will define the networks shown in Figure 10, and the corresponding output file, can be obtained from the course website along with this assignment.

    When you are certain your program is correct, save you main function in a file named

    ```
    ece0301_inclass12_step13.cpp
    ```

    Submit this file, and all of the following files, to the class repository.

    ```
    Node.h
    Node.cpp
    Component.h
    ```

```
Component.cpp
Network.h
Network.cpp
DCVoltageSource.h
DCVoltageSource.cpp
DCCurrentSource.h
DCCurrentSource.cpp
Resistor.h
Resistor.cpp
VoltageDivider.h
VoltageDivider.cpp
CurrentDivider.h
CurrentDivider.cpp
```

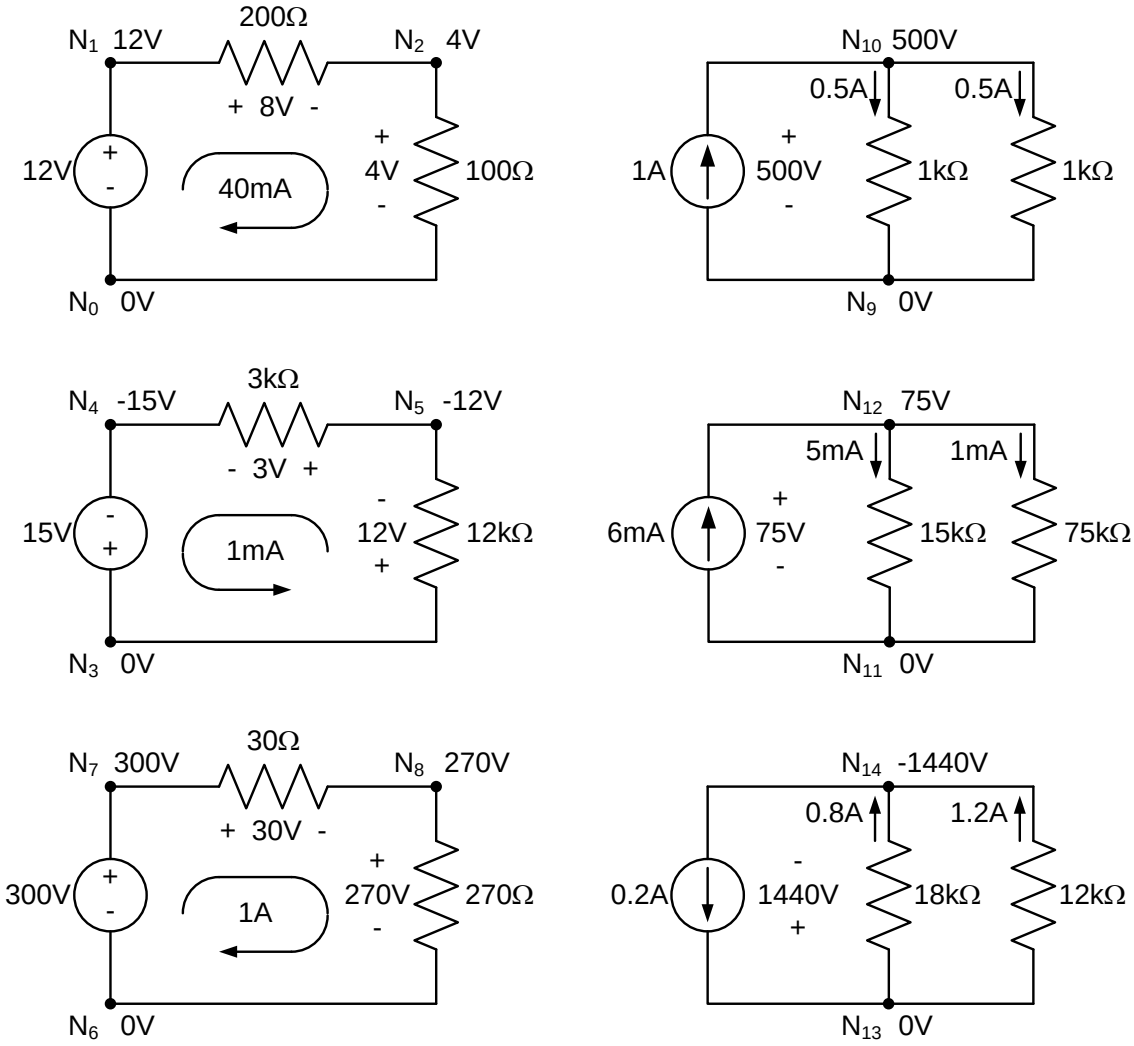

Figure 10: Circuits to simulate