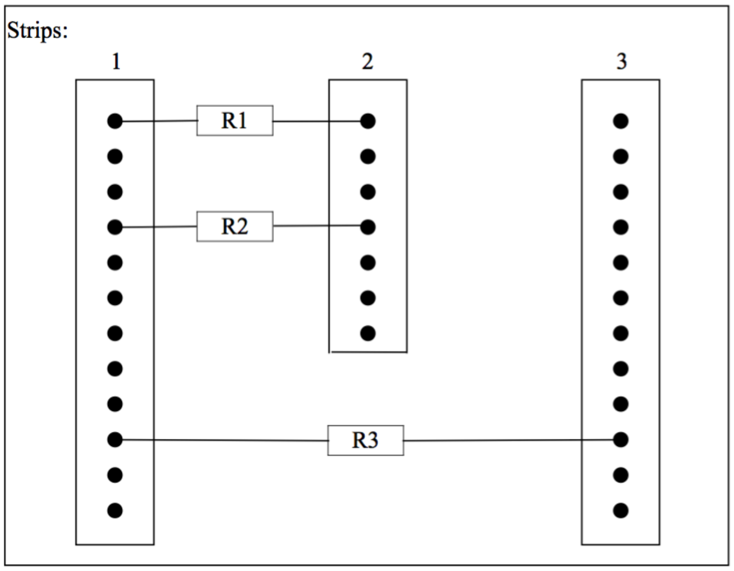
Lab Assignment #4: Introduction to Objects

1. Objectives

The objective of this assignment is to provide you with a practical introduction to programming with C++ objects. This assignment requires you to complete the implementation of two simple classes and their associated member functions, as well as performing basic I/O operations.

2. Problem Statement

Your task is to implement a simple program for managing resistors in a prototype circuit. A prototype circuit board has a number of fixed ***strips,*** and ***resistors*** are connected to two different strips of the prototype board. Each resistor has certain values associated with it, e.g., its resistance value. Moreover, recorded with each resister are the two strips the resistor is connected to, which we will arbitrarily call leftStrip and rightStrip.



The figure above depicts three strips and three resistors. Resistor R1 is connected to Strip 1 and Strip 2, as is resistor R2. The resistor R3 is connected to Strip 1 and Strip 3.

You are to implement two classes, as described below. For **this assignment only,** you are free to add new members to the classes, if required, but you must not change or remove any of the class elements that we have provided (e.g., you cannot change any of the provided members from "private" to "public", change their data types, parameter lists, etc.). Any variables that you add to a class must be "private"; if you need to access the variables outside of your class, you should write "public" functions to read them or change them.

The Resistor Class

A *Resistor* object exists for each resistor used. The object contains the resistor's resistance level, its name, and the index of the two strips it is connected to. Its declaration is provided to you in Resistor.h, and you should implement the class in Resistor.cpp, along with any additional member functions that you define.

```
class Resistor
{
  private:
      double          resistance; // resistance (in Ohms)
      char*           name;       // a C-style string
      int             leftStrip;
      int             rightStrip;


  public:
      Resistor ();
      ~Resistor ();

      bool            set_name (char* _name);
      bool            set_resistance (double _resistance);
      void            print ();
};
```

The Strip Class

Each *Strip* object contains an array of *Resistor* objects large enough to hold at most a total of (MAX_RESISTORS_PER_STRIP) entries. The class declaration is provided to you in Strip.h, and you should implement the class in Strip.cpp, along with any additional member functions that you define.

```
class Strip
{
  private:
      int             number;
      Resistor        resistors[MAX_RESISTORS_PER_STRIP];

  public:
        Strip ();
        ~Strip ();

        bool         addResistor (double resistance,
                            char* name, int leftStrip,
                            int rightStrip);
        void         print ();
};
```

The Strip::addResistor function is used to populate the resistors array. You may find it necessary to expand the Strip class (e.g., in order to keep track of the number of resistors you have added). A resistor is added only to the strip identified by the resistor's leftStrip value, and leftStrip must be less than rightStrip (if not, you should swap the values before inserting).

3. Input and Output

The lab builds on the Driver.cpp that you wrote for the previous lab assignment. As with Lab #3, all output and error messages should be printed to standard output (cout), and all input should be read from standard input (cin). Your program should accept the following two commands:

insert [*resistance*] [*node1*] [*node2*] [*name*]

Parameters

`resistance`

A positive, floating-point number.

`node1 and node2`

Both values must be positive integers in the range between 0 and MAX_STRIP_NUMBER (which is defined for you in Strip.h).

`Name`

A string. To simplify your error checking, you can assume that we will not test your program with a string longer than the number of characters specified in MAX_RESISTOR_NAME_LEN (which is defined for you in the Resistor.h).

Action

Your program must add each inserted resistor into a Strip object that corresponds to the leftStrip of the resistor. The leftStrip should be the lower value of node1 and node2, and rightStrip should be the larger of node1 and node2. For example, if the user enters:

```
insert 100 2 1 R1
```

then leftStrip should be 1and rightStrip should be 2. The command should return the "success" and "error" output described below.

print [strip]

Parameters

`strip`

This parameter can either be a positive integer or the string "all". If it is an integer, it must be in the range between 0 and MAX_STRIP_NUMBER (which is defined for you in Strip.h).

Action

If a specific strip number is specified, then the command should print the specified strip, otherwise it should print all strips.

For each strip, a report similar to the following should be printed, listing each resistor (if any) that has this strip as a leftStrip:

```
Strip #1
--- Resistor R1: 222.2 ohms, strip 1 -> strip 2
--- Resistor R2: 333 ohms, strip 1 -> strip 3
```

No output should be produced if there are no resistors to list.

The command should return the "success" and "error" output, which is described below. It should print nothing other than "`print: success`" if there are no resistors connected to the strip(s).

As in Assignment #3, the input will be terminated with an end of file (eof).

The following is an example of input commands and output messages. Input fields are shown in **bold,** while output and error messages are in normal text:

```
insert 111 -5 16 R0
Error: argument is not a valid number.
insert 222.2 1 2 R1
insert: success.
insert 333 1 3 R2
insert: success.
insert 444.4 2 3 R3
insert: success.
print all
Strip #1
--- Resistor R1: 222.2 ohms, strip 1 -> strip 2
--- Resistor R2: 333 ohms, strip 1 -> strip 3
Strip #2
--- Resistor R3: 444.4 ohms, strip 2 -> strip 3
print: success.
print 1
Strip #1
--- Resistor R1: 222.2 ohms, strip 1 -> strip 2
--- Resistor R2: 333 ohms, strip 1 -> strip 3
print: success.
print 2
Strip #2--- Resistor R3: 444.4 ohms, strip 2 -> strip 3
print: success.
print 3
print: success.
(eof)
```

The purpose of this assignment is to test your understanding of C++ classes. As such, you can assume that we will not test your code with fewer than the specified number of parameters for any given command. You may also assume that there will be only one command per line, that the individual commands will not be split into multiple lines, and that we will only test “insert” and “print” commands.

4. Procedure

Create a sub-directory called `lab4` in your `ece244` directory, using the `mkdir` command, and protect it with the `chmod` command. Make it your working directory, and then download the Lab 4 source files that are provided on the course website. Copy your `Driver.cpp` from Lab #3 into this directory, and remove or rename its `main()` function. Using these files as a starting point, generate a solution to this assignment. Make a new main function in a file named “`main.cpp`”, and make appropriate function call(s) from it to accept and respond appropriately to the input commands. Additionally, create a `Makefile` that will produce a lab4 executable when you type: “`make`”.

5. Deliverables

Your lab submission should consist of only the following eight files:

main.cpp – Contains the main() function for your program.

Driver.cpp – A version of your code from Lab #3, modified to interpret and respond to the “insert” and “print” commands described above.

Driver.h – Contains the declaration for your function(s) in Driver.cpp, so that they can be called from main.cpp

Resistor.h – Contains the declaration for the Resistor class, including any additions that you have made.

Resistor.cpp – Contains the implementation of all member functions for the Resistor class.

Strip.h – Contains the declaration for the Strip class, including any additions that you have made.

Strip.cpp – Contains the implementation of all member functions for the Strip class.

Makefile – A makefile that will compile your code and produce an executable called “lab4” when the “make” command is entered. (The default behaviour of your `Makefile` should only compile the program when “make” is run: it should not run the program or perform any other tasks.)

Please note that it is **ESSENTIAL** that your Makefile produce an executable called lab4 when it is run, and that your lab4 accept input and produce output **EXACTLY** as specified in the assignment description, without adding any additional output. Please submit your files by using the usual “submitece244f” command:

```
    submitece244f 4 main.cpp Driver.h Driver.cpp Strip.h \
               Strip.cpp Resistor.h Resistor.cpp Makefile
```

Before submitting, you should use the usual “`exercise`” and “`chksubmit`” commands to verify that your submission is working as expected.