ECE 3220 Lab7
String, Vectors and Operators in C++

Marshall Lindsay
mblgh6@mail.missouri.edu
14211662

3/21/2017

Objective:

The goal for lab7 was to introduce the concept of overloading functions as well as the use of vectors.

Discussion:

This lab was essentially an extension on Lab6. We changed our data signal arrays to vectors, created overloaded operators to work with our signal class, and created a non-member overloaded operator to add two signals together. What was interesting from this lab came from the observations when trying to implement the new overloaded operators. Because we used vectors and now had a proper operator implementing the other functions was incredibly easy and straightforward. Answered to the posed questions in the code was answered in the code.

Experiments and Results:

By using strings as often as possible an increase of proper exception handling was obtained. This was due to the fact that entire lines of input was gathered at a time, operated on and checked for validity. This created a input-bug free program. Screenshots of outputs can be found in the Figures section.

Conclusion:

The objective of this lab was to become more familiar with the concept of overloading functions as well as the use of vectors and strings. By the end of this lab a concrete understanding of each of these topics was obtained.

Figures:

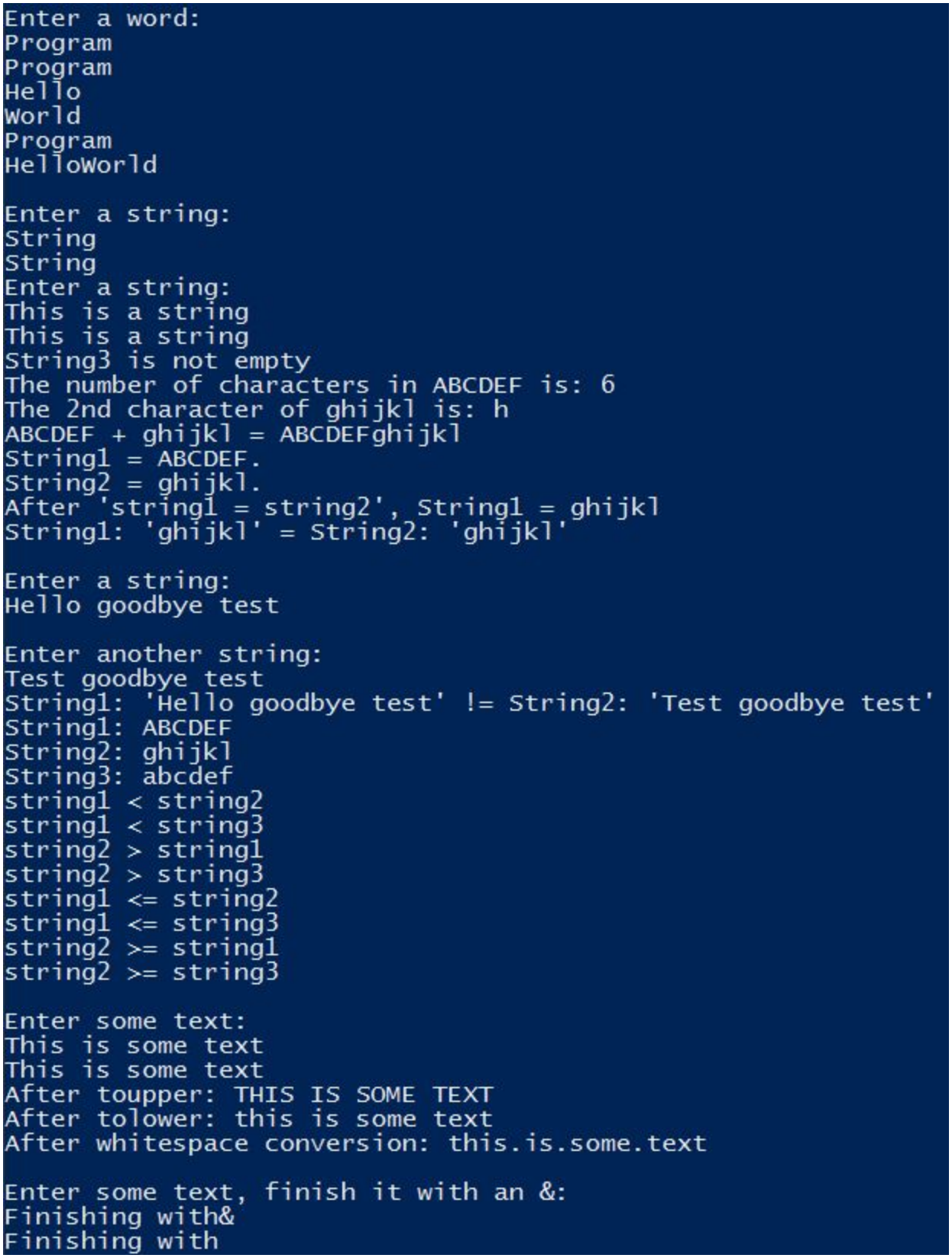

Figure1: Screenshot showing the running of Lab7_strings.cpp

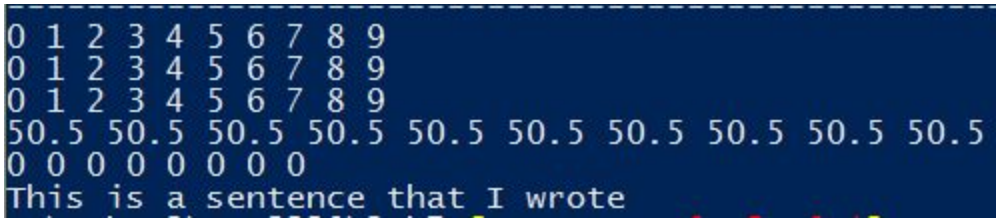

Figure2: Screenshot showing the execution of Lab7_vectors.cpp

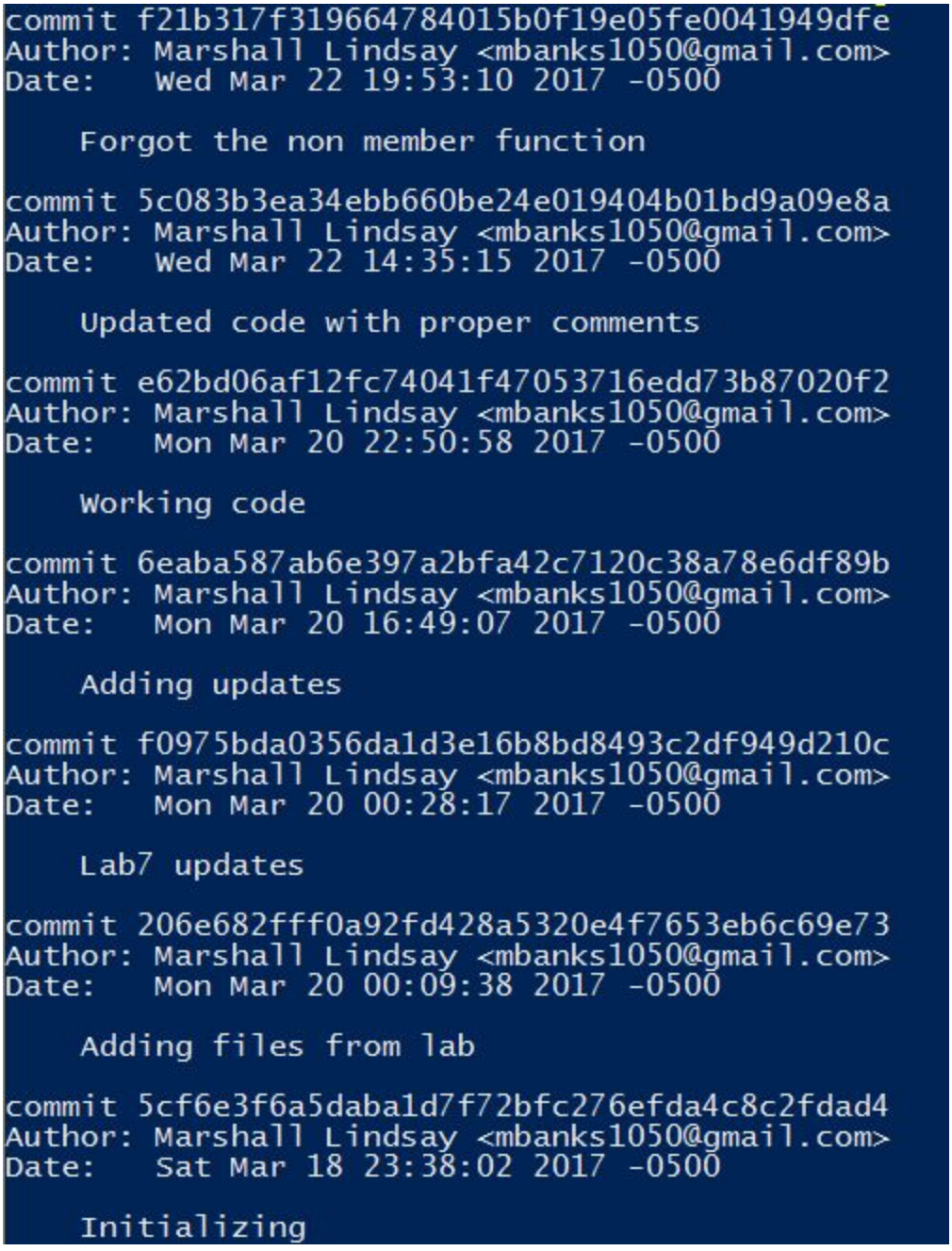

Figure3: Screenshot showing GitHub commits

Code:

Lab7_Vectors.cpp

```
// Lab7_vectors.cpp
// This program shows some simple vector examples, and asks for simple tasks.
// Reference: Lippman, section 3.3

// Author: Luis Rivera

// IMPORTANT NOTE: YOU MAY NEED TO COMPILE ADDING THE FOLLOWING FLAG: -std=c++11
//                 OR: -std=c++0x
// Example:  g++ Lab7_vectors.cpp -o Lab7_vectors -std=c++11
//      or:  g++ Lab7_vectors.cpp -o Lab7_vectors -std=c++0x
// Some initialization methods and other things are not supported by the old standard.

#include <iostream>
#include <vector>

using namespace std;

// Main function. Shows a few examples. Complete the assignments stated in the comments.
int main()
{
        int i;
        vector<int> ivec1(5), ivec2;
        vector<double> dvec1{5.1}, dvec2(5,1.5);
        vector<string> svec1 = {"hello", "world"};
        // vector<myClass> myCvec;          // you can have vectors of objects
        // in general: vector<Type> vec;      // vector is a template

        for(int i = 0; i < ivec1.size(); i++)
                cout << ivec1[i] << endl;
        cout << "\n----------------------------------------------------------------" << endl;

        for(auto i:ivec1)    // This is equivalent to the above for loop
                cout << i << endl;
        cout << "\n----------------------------------------------------------------" << endl;

        for(auto i:dvec1)
                cout << i << endl;
        cout << "\n----------------------------------------------------------------" << endl;

        for(auto i:dvec2)
                cout << i << endl;
        cout << "\n----------------------------------------------------------------" << endl;

        for(auto i:svec1)
                cout << i << endl;
        cout << "\n----------------------------------------------------------------" << endl;

        cout << "Original size: " << ivec2.size() << endl;
        ivec2.push_back(50);
        cout << "New size: " << ivec2.size() << "\nAdded element: " << ivec2[0] << endl;
```

```
        cout << "\n----------------------------------------------------------------" << endl;

// ************************************************************************
// Try all the ways to initializa a vector shown in Table 3.4. Use the
// vectors above and/or declare new vectors.
// Some of those operations have already been used, but write your
// own examples.

// Do exercises 3.14 and 3.15 from Lippman (page 102)

// Try all the vector operations shown in table 3.5. Use the vectors above
// or define new ones. Try different types.
// ************************************************************************

        vector<int> scores(10);//1
        for( i = 0; i < scores.size(); i++ ){
                scores[i] = i;
        }

        for(i = 0; i < scores.size(); i++){
                cout<< scores[i] <<" ";
        }
        cout<<endl;

        vector<int> copy(scores);//2

        for(i = 0; i < copy.size(); i++){
                cout<< copy[i] <<" ";
        }
        cout<<endl;

        vector<int> copy2 = scores;//3


        for(i = 0; i < copy2.size(); i++){
                cout<< copy2[i] <<" ";
        }
        cout<<endl;

        vector<double> percentages(10,50.5);//4

        for(auto i:percentages){
                cout<< i <<" ";
        }
        cout<<endl;

        int number = 8;

        vector<int> foo(number);//5
        for(auto i:foo){
                cout<< i <<" ";
        }
        cout<<endl;
```

```
	vector<string> sentence{"This", "is", "a", "sentence that I wrote"};//6 and 7
	for(auto i:sentence){
		cout<< i <<" ";
	}
	cout<<endl;


	return 0;
}
```

Lab7_Strings.cpp

```
// Lab7_strings.cpp
// This program shows some simple string examples, and asks for simple tasks.
// Reference: Lippman, section 3.2

// Author: Luis Rivera

// IMPORTANT NOTE: YOU MAY NEED TO COMPILE ADDING THE FOLLOWING FLAG: -std=c++11
//		OR: -std=c++0x
// Example:  g++ Lab7_strings.cpp -o Lab7_strings -std=c++11
//      or:  g++ Lab7_strings.cpp -o Lab7_strings -std=c++0x
// Some initialization methods and other things are not supported by the old standard.

#include <iostream>
#include <string>

using namespace std;

// Main function. Shows a few examples. Complete the assignments stated in the comments.
int main()
{
	string s1, s2("Hello"), s3 = "World";
	cout << "\nEnter a word:" << endl;
	cin >> s1;
	cin.ignore();		// to consume the '\n' character. Otherwise, you may get issues with
					// the getline() below. Try commenting this out. Any issues?
					// Yes. STDIN still contained the /n from out input. This messed with
things.

	string s4(s1);

	cout << s1 << endl
		<< s2 << endl
		<< s3 << endl
		<< s4 << endl;

	s1 = s2 + s3;
	cout << s1 << endl;

// ************************************************************************
// Try all the operations in Table 3.2 using the strings above and
// using new strings that you may declare.
// Some of those operations have already been used, but write your
```

```
// own examples.
// ***************************************************************************
	string string1, string2,string3;
	string1 = "ABCDEF";
	string2 = "ghijkl";
	//1 and 2
	cout<<"\nEnter a string:"<<endl;
	cin>>string3;
	cout<<string3;
	//3
	cin.ignore();
	cout<<"\nEnter a string:"<<endl;
	getline(cin, string3);
	cout<<string3;
	//4
	if(!string3.empty()){
		cout<<"\nString3 is not empty"<<endl;
	}else{
		cout<<"\nString3 is empty"<<endl;
	}
	//5
	cout<<"The number of characters in "<<string1<<" is: "<<string1.size()<<endl;
	//6
	cout<<"The 2nd character of "<<string2<<" is: "<<string2[1]<<endl;
	//7
	cout<<string1<<" + "<<string2<< " = "<<string1 + string2<<endl;
	//8

	cout<<"String1 = "<<string1<<"."<<"\nString2 = "<<string2<<"."<<endl;
	string1 = string2;
	cout<<"After 'string1 = string2', String1 = "<<string1<<endl;
	//9
	if(string1 == string2){
		cout<<"String1: '"<<string1<<"' = String2: '"<<string2<<"'"<<endl;
	}
	//10
	cout<<"\nEnter a string: "<<endl;
	getline(cin, string1);
	cout<<"\nEnter another string: "<<endl;
	getline(cin, string2);
	if(string1 != string2){
		cout<<"String1: '"<<string1<<"' != String2: '"<<string2<<"'"<<endl;
	}else{
		cout<<"String1: '"<<string1<<"' = String2: '"<<string2<<"'"<<endl;
	}

	//11

	string1 = "ABCDEF";
	string2 = "ghijkl";
	string3 = "abcdef";
	cout<<"String1: "<<string1<<endl;
	cout<<"String2: "<<string2<<endl;
	cout<<"String3: "<<string3<<endl;
```

```
	if(string1 > string2){
		cout<<"string1 > string2"<<endl;
	}else if(string1 < string2){
		cout<<"string1 < string2"<<endl;
	}
	if(string1 > string3){
		cout<<"string1 > string3"<<endl;
	}else if(string1 < string3){
		cout<<"string1 < string3"<<endl;
	}
	if(string2 > string1){
		cout<<"string2 > string1"<<endl;
	}else if(string2 < string1){
		cout<<"string2 < string1"<<endl;
	}
	if(string2 > string3){
		cout<<"string2 > string3"<<endl;
	}else if(string2 < string3){
		cout<<"string2 < string3"<<endl;
	}
	/**********************************/
	if(string1 >= string2){
		cout<<"string1 >= string2"<<endl;
	}else if(string1 <= string2){
		cout<<"string1 <= string2"<<endl;
	}
	if(string1 >= string3){
		cout<<"string1 >= string3"<<endl;
	}else if(string1 <= string3){
		cout<<"string1 <= string3"<<endl;
	}
	if(string2 >= string1){
		cout<<"string2 >= string1"<<endl;
	}else if(string2 <= string1){
		cout<<"string2 <= string1"<<endl;
	}
	if(string2 >= string3){
		cout<<"string2 >= string3"<<endl;
	}else if(string2 <= string3){
		cout<<"string2 <= string3"<<endl;
	}


// ------------------------------------------------------------------------


// ************************************************************************
// Use a "Range for" (Lippman, page 93) and operations in table 3.3 to:
// 1) change the case of the letters in your input line above (upper to
//    lowercase and vice-versa).
// 2) Replace any whitespace with a dot ('.').
// Print the converted text.
// ************************************************************************
```

```
	string line;
	cout << "\nEnter some text:" << endl;
	getline(cin, line);
	cout << line << endl;

	for(auto &c : line){
		c = toupper(c);
	}
	cout<<"After toupper: "<<line<<endl;
	for(auto &c : line){
		c = tolower(c);
	}
	cout<<"After tolower: "<<line<<endl;

	for(auto &c : line){
		if(isspace(c)){
			c = '.';
		}
	}
	cout<< "After whitespace conversion: "<<line<<endl;
	cout << "\nEnter some text, finish it with an &:" << endl;
	getline(cin, line, '&');		// the delimiter can be any character
	cout << line << endl;

	return 0;
}
```

Lab7_Operators.cpp

```
// Lab7_operators.cpp
// This program provides a few examples about operators in C++
// Study the code, run it, pay attention to the comments and questions
// posted, and use the ideas to complete the assignments described in
// the lab 7 guide.
// Reference: Lippman, sections 14.1 - 14.4

// Author: Luis Rivera

// IMPORTANT NOTE: YOU MAY NEED TO COMPILE ADDING THE FOLLOWING FLAG: -std=c++11
//		OR: -std=c++0x
// Example:  g++ Lab7_operators.cpp -o Lab7_operators -std=c++11
//	or:  g++ Lab7_operators.cpp -o Lab7_operators -std=c++0x
// Some initialization methods and other things are not supported by the old standard.

#include <iostream>

using namespace std;

class Lab7
{
public:
	int a[2]; // could be private, but methods whould be needed to access this member.
```

```
	// operators as member functions: 'this' is bound to the left hand operand
	bool operator>(const Lab7 &x) const;
	void operator+(const Lab7 &x);
	void operator*(int factor);
	void Printvals();
	// constructor(s)
	// destructor
};

// ---------- Member functions ---------------------------------
bool Lab7::operator>(const Lab7 &x) const
{
	if ((a[0] > x.a[0]) & (a[1] > x.a[1]))
		return 1;
	else
		return 0;
}

void Lab7::operator+(const Lab7 &x)
{
	a[0] += x.a[0];
	a[1] += x.a[1];
}

void Lab7::operator*(int factor)
{
	a[0] *= factor;
	a[1] *= factor;
}

void Lab7::Printvals()
{
	cout << "\na[0] = " << a[0] << ", a[1] = " << a[1] << endl;
}

// -------- Operators can be Non-member functions -------------------
Lab7 operator+(const Lab7 &lhs, const Lab7 &rhs)
{
	Lab7 sum;
	sum.a[0] = lhs.a[0] + rhs.a[0];
	sum.a[1] = lhs.a[1] + rhs.a[1];

	return sum;
}

// Main function. Shows a few examples about using the operators.
int main()
{
	Lab7 obj1, obj2, obj3;
	int f = 10;

	obj1.a[0] = 1; obj1.a[1] = 5;
	obj2.a[0] = 10; obj2.a[1] = 20;
```

```
	if(obj1 > obj2)	// normal expression. Which operator is being called? The member function one
		cout << "\nObject 1 is bigger than object 2" << endl;
	else
		cout << "\nObject 1 is not bigger than object 2" << endl;

	 if(obj1.operator>(obj2))	// equivalent function call (eq. to obj1 > obj2)
		cout << "\nObject 1 is bigger than object 2" << endl;
	else
		cout << "\nObject 1 is not bigger than object 2" << endl;


	if(obj2 > obj1)	// Which operator is being called? The member function one
		cout << "\nObject 2 is bigger than object 1" << endl;
	else
		cout << "\nObject 2 is not bigger than object 1" << endl;

	if(obj2.operator>(obj1))	// explicit function call, equivalent to the previous
		cout << "\nObject 2 is bigger than object 1" << endl;
	else
		cout << "\nObject 2 is not bigger than object 1" << endl;

	cout << "-----------------------------------------------------------------" << endl;
	obj1.Printvals();
	obj2.Printvals();

	obj1 + obj2;		// normal expression. Which operator is being called? The member function
	// obj1.operator+(obj2);	// equivalent, try it out
	obj1.Printvals();
	obj2.Printvals();

	obj2 + obj1;		// normal expression. Which operator is being called? The member function
	// obj2.operator+(obj1);	// equivalent, try it out
	obj1.Printvals();
	obj2.Printvals();

	cout << "-----------------------------------------------------------------" << endl;
	obj1*2;		// Would 2*obj1 work? Give it a try.
	obj2*f;		// Would f*obj2 work? Give it a try.
	//2*obj1;
	obj1.Printvals();
	obj2.Printvals();

	cout << "-----------------------------------------------------------------" << endl;
	// Which operator is being called next?
	//obj3 = operator+(obj1, obj2);	// Would  obj3 = obj2 + obj1; work?
	obj3.a[0] = 0;
	obj3.a[1] = 0;
	obj3 + obj1;
	obj3 + obj2;
	obj1.Printvals();				// If not, how could you make it work? By changing what the
member function
	obj2.Printvals();				//recieves and returns;
	obj3.Printvals();
```

```
	obj3 = operator+(obj2, obj1);		// is this the same as before? This is the non member function
	obj1.Printvals();
	obj2.Printvals();
	obj3.Printvals();

	return 0;
}
```

Lab7_Excersizes.cpp

```
/*
	Marshall Lindsay
	3/20/17
	Lab7
	Excersizes 3.14 and 3.15

	3.14 - Write a program to read a sequence of ints from cin and store those
	values in a vector;
	3.15 - Repeat the previous program but read strings this timee;

*/


#include <iostream>
#include <vector>
#include <string>

using namespace std;

int main(){

	int input;
	string input2;
	int i;
	vector<int> integers;
	vector<string> strings;

	cout<<"Please enter 10 integers: "<<endl;

	for(int i = 0; i < 10; i++){
		cin>> input;
		integers.push_back(input);
	}

	for(auto i:integers){
		cout<< i << " ";
	}
	cout<<endl;

	cout<<"Please enter 5 strings: "<<endl;
	cin.ignore();
	for(int i = 0; i < 5; i++){
		getline(cin, input2);
		strings.push_back(input2);
```

```
	}

	for(auto i:strings){
		cout<< i << " ";
	}


	return(0);
}
```

Lab7.cpp

```
/*
	Marshall Lindsay
	Lab7
	3/20/17
*/

#include <string>
#include <iostream>
#include <fstream>
#include <vector>
#include <locale>

using namespace std;

#define DEFAULT_DATA_FILE "defaultData.txt"

//Signal Class
class Signal{
	private:
		string fileName;
		int length;
		double max_value;
		double average;
		vector<double> data;
		void calcAvg();
		double calcMaxValue();

	public:

		void operator*(double x);
		void operator+(double x);
		void addData(double x, int i);
		void setName(string);
		double getData(int);
		void setAvg(double);
		double getAvg();
		void setData(double, int);
		double getMax();
		void setMax(double);
		int getLength();
		void center();
		void normalize();
```

```
                    void Sig_info();
                    void Save_file(string);
                    Signal();
                    Signal(string L);
                    Signal(const char* fileName);
                    ~Signal();

};

//Default constructor. Sets all private members to zero
Signal::Signal(){
        this -> fileName = "void";
        this -> length = 0;
        this -> max_value = 0;
        this -> average = 0;

}

//Filenumber constructor. Opens the data file with the corresponding filenumber
//and sets all the private members.
Signal::Signal(string L){
        fstream dataFile;
        string fileName;

        //Declare a const char* to be used with the .open() on line 115
        //that points to the file name.
        const char* ptrFileName = fileName.c_str();


        //Append the integers into the file name template
        fileName = std::string("Raw_data_") + L + ".txt";

        this->fileName = fileName;
        //Open the correct file
        dataFile.open(ptrFileName);

        //Check that the file was opened correctly.
        //If not, set default values and return
        if(!dataFile.is_open()){
                cout<<"\nCould not open "<<fileName<<" setting default values!"<<endl;
                this -> fileName = "NULL";
                this -> length = 0;
                this -> max_value = 0;
                this -> average = 0;
                return;
        }

        //Read the number of integers in the datafile
        dataFile >>  this -> length;
        vector<double> copy(this -> length, 0);
        //Allocate memory for the data array.
        this -> data = copy;

        //Read the max value from the data file
```

```
	dataFile >> this -> max_value;

	//Read the data from the file and place in the data array.
	for(int i = 0; i < this -> data.size(); i++){
		dataFile >> this -> data[i];
	}
	//Close the file
	dataFile.close();
	//Calculate the average value from the data
	calcAvg();
	return;
}

Signal::Signal(const char* fileName){
	fstream dataFile;

	this->fileName = fileName;
	//Open the correct file
	dataFile.open(fileName);

	//Check that the file was opened correctly.
	//If not, set default values and return
	if(!dataFile.is_open()){
		cout<<"\nCould not open "<<fileName<<" setting default values!"<<endl;
		this -> fileName = "NULL";
		this -> length = 0;
		this -> max_value = 0;
		this -> average = 0;
		return;
	}

	//Read the number of integers in the datafile
	dataFile >>  this -> length;
	vector<double> copy(this -> length, 0);
	//Allocate memory for the data array.
	this -> data = copy;

	//Read the max value from the data file
	dataFile >> this -> max_value;

	//Read the data from the file and place in the data array.
	for(int i = 0; i < this -> data.size(); i++){
		dataFile >> this -> data[i];
	}

	dataFile.close();
	calcAvg();
	return;
}

//Destructor. Doesn't do anything really. It prints to know it was called.
Signal::~Signal(){
	cout<<"Destructor"<<endl;
}
```

```
double Signal::getAvg(){
	return(this-> average);
}

void Signal::setAvg(double value){
	this->average = value;
}

void Signal::setName(string name){
	this->fileName = name;
}

//Finds the maximum value in the data set and returns it.
double Signal::calcMaxValue(){
	double hold = 0;

	for(int i = 0; i < this -> data.size(); i++){
		if(this -> data[i] > hold)
			hold = this -> data[i];
	}

	return(hold);

}

void Signal::setMax(double value){
	this->max_value = value;
}

//Returns the value at the specified location in the data vector
double Signal::getData(int i){
	if(i < (this -> length)-1)
		return(this->data[i]);
	return(0);
}

//Returns the objects length member
int Signal::getLength(){
	return(this->length);
}

//Calculates the average of the data set and sets the average member.
void Signal::calcAvg(){

	double total = 0;

	for(int i = 0; i < this -> data.size(); i++){
		total += this -> data[i];
	}

	this -> average = (double)(total) / this -> length;

}
```

```
//Sets the data at the index to the value specified.
void Signal::setData(double value, int i){
	this->data[i] = value;
}
//Returns the max_value
double Signal::getMax(){
	return(this->max_value);
}

//Overloaded operator to multiply a signal by a double. Multiplies each
//data value by the double value sent to the function. Finds the new max
//value and calculates the new average.
void Signal::operator*(double x){

	for(int i = 0; i < this -> length; i++){
		this -> data[i] = this -> data[i] * x;
	}
	this -> max_value = calcMaxValue();
	calcAvg();
}

//Overloaded operator to add a double to a signal. Adds the double to each
//of the data values. Finds the new max value and calulates the new average.
void Signal::operator+(double x){

	for(int i = 0; i < this -> length; i++){
		this -> data[i] = this -> data[i] + x;
	}
	this -> max_value = calcMaxValue();
	calcAvg();
}

//Adds a value x to the data indexed by i
void Signal::addData(double x, int i){
	this->data[i] += x;
}

//Centers the data by using the overloaded addition operator.
void Signal::center(){
	this -> operator+(-(this -> average));
	this -> max_value = calcMaxValue();
	calcAvg();
}

//Normalizes the data by using the overloaded multiplication operator.
void Signal::normalize(){
	operator*(1/(this->max_value));
	this -> max_value = calcMaxValue();
	calcAvg();

}

//Prints all of the information about the signal. This was changed from
```

```
//Lab6 to include the printing of the data and the filename where the data
//came from.
void Signal::Sig_info(){
	cout<<"\nData file name: "<< this->fileName
	<<"\nNumber of data points (length): "<<this->length
	<<"\nMaximum value (max_value): "<<this->max_value
	<<"\nAverage of data (average): "<<this->average<<endl;
	for( int i = 0; i < this->length; i++){
		cout<< this ->data[i] << endl;
	}
}

//Saves the current signal to a new file whose name is that
//of the string that is passed to the function.
void Signal::Save_file(string newFileName){
	newFileName = std::string(newFileName) + ".txt";
	const char* newFilePtr = newFileName.c_str();

	//open save file

	ofstream saveFile(newFilePtr);

	//Save the Length, Max Value, and the data to the save file
	saveFile << this->length<< " " << this->max_value<<endl;

	for(auto i:data){
		saveFile<< i <<endl;
	}

	//Close the file
	saveFile.close();
	return;
}
Signal operator+(Signal, Signal);
void optionMenu();
double scaling();
double offsetting();
Signal addSignals();
string fileSave();

int main(){

	locale loc; //Used for some error checking.
	string userInput;
	string fileName;
	const char* fileNamePtr = fileName.c_str();
	Signal* dataSignal;

	cout<<"\n****Lab7****"<<endl;
	while(1){

		cout<<"\nEnter 'F' to use the file name or 'N' to use the file number:"<<endl;
		getline(cin, userInput);
		//Convert input to upper case
```

```
for(auto &c : userInput){
	c = toupper(c);
}
//Error check input
while(userInput != "F" && userInput != "N"){
	cout<<"\nInvalid input!"<<endl;
	cout<<"\nEnter 'F' to use the file name or 'N' to use the file number:"<<endl;
	getline(cin, userInput);
	for(auto &c : userInput){
	c = toupper(c);
	}
}
//Logic on input
if(userInput == "F"){
	cout<<"Please enter the file name to be opened:"<<endl;
	getline(cin, fileName);
	dataSignal = new Signal(fileNamePtr);
}else{
	cout<<"\nPlease enter the file number to be opened:"<<endl;
	getline(cin, userInput);
	//Error check on file number input
	while(userInput.size() > 2 || (!isdigit(userInput[0],loc) && !isdigit(userInput[1],loc))){
		cout<<"\nInvalid file number!"<<endl;
		cout<<"\nPlease enter the file number to be opened:"<<endl;
		getline(cin, userInput);
	}
	if(userInput.size() == 1){
		string hold = userInput;
		userInput = "0";
		userInput = userInput + hold;
	}
	dataSignal = new Signal(userInput);
}


while(1){
	//Display option menu
	optionMenu();

	//Get user choice
	getline(cin, userInput);

	for(auto &c: userInput){
		c = toupper(c);
	}
	//Logic on the user input for operations
	if(userInput == "S"){
		dataSignal->operator*(scaling());
	}else if(userInput == "O"){
		dataSignal->operator+(offsetting());
	}else if(userInput == "P"){
		dataSignal->Sig_info();
	}else if(userInput == "C"){
		dataSignal->Sig_info();
```

```
                }else if(userInput == "N"){
                        dataSignal->normalize();
                }else if(userInput == "V"){
                        dataSignal->Save_file(fileSave());
                }else if(userInput == "Q"){
                        break;
                }else if(userInput == "A"){
                        *dataSignal = *dataSignal + addSignals();
                }else{
                        cout<<"Invalid input!"<<endl;
                }
        }

        cout<<"Would you like to manipulate another signal?(Y N)"<<endl;
        getline(cin, userInput);
        for(auto &c: userInput){
                c = toupper(c);
        }
        while(userInput != "Y" && userInput != "N"){
                cout<<"\nInvalid input!"<<endl;
                cout<<"Would you like to manipulate another signal?(Y N)"<<endl;
                getline(cin, userInput);
                for(auto &c:userInput){
                        c = toupper(c);
                }
        }
        if(userInput == "N"){
                delete dataSignal;
                break;
        }

        delete dataSignal;

    }

    return(0);
}

void optionMenu(){

    cout<<"What would you like to do with the signal?"<<endl;
    cout<<"(S) : Scale the data\n"
        <<"(O) : Offset the data\n"
        <<"(P) : Print statistics for the data\n"
        <<"(C) : Center the data\n"
        <<"(N) : Normalize the data\n"
        <<"(A) : Add signals\n"
        <<"(V) : Save data to file\n"
        <<"(Q) : Quit"<<endl;
    return;
}

double scaling(){
    double value;
```

```
	cout<<"Please enter the value you wish to scale the data by:"<<endl;
	cin >> value;

	//Check that the user input is a double type
	while(cin.fail()){
		cin.clear();
		fflush(stdin);
		cout<<"Invalid input! Please enter a number!"<<endl;
		cin >> value;
	}
	fflush(stdin);
	return(value);

}

double offsetting(){
	double value;

	cout<<"Please enter the value you wish to offset the data by:"<<endl;
	cin >> value;

	//Check that the user input is a double type
	while(cin.fail()){
		cin.clear();
		fflush(stdin);
		cout<<"Invalid input! Please enter a number!"<<endl;
		cin >> value;
	}
	fflush(stdin);
	return(value);
}

string fileSave(){

	string newFileName;

	cout<<"Please enter the name of the file to save the updated data to:"<<endl;
	getline(cin, newFileName);

	return(newFileName);

}

//Opens a new signal to be added to the current one
//Returns that signal.
Signal addSignals(){
	locale loc;
	string userInput;
	Signal* dataSignal;
	string fileName;
	const char* fileNamePtr = fileName.c_str();
	cout<<"\nPlease specify which data file contains the signal you wish to add."<<endl;
	cout<<"Enter 'F' to use the file name or 'N' to use the file number:"<<endl;
```

```
		getline(cin, userInput);
		//Convert input to upper case
		for(auto &c : userInput){
			c = toupper(c);
		}
		//Error check input
		while(userInput != "F" && userInput != "N"){
			cout<<"\nInvalid input!"<<endl;
			cout<<"\nEnter 'F' to use the file name or 'N' to use the file number:"<<endl;
			getline(cin, userInput);
			for(auto &c : userInput){
			c = toupper(c);
			}
		}
		//Logic on input
		if(userInput == "F"){
			cout<<"Please enter the file name to be opened:"<<endl;
			getline(cin, fileName);
			dataSignal = new Signal(fileNamePtr);
		}else{
			cout<<"\nPlease enter the file number to be opened:"<<endl;
			getline(cin, userInput);
			//Error check on file number input
			while(userInput.size() > 2 || (!isdigit(userInput[0],loc) && !isdigit(userInput[1],loc))){
				cout<<"\nInvalid file number!"<<endl;
				cout<<"\nPlease enter the file number to be opened:"<<endl;
				getline(cin, userInput);
			}
			if(userInput.size() == 1){
				string hold = userInput;
				userInput = "0";
				userInput = userInput + hold;
			}
			dataSignal = new Signal(userInput);
		}

return(*dataSignal);
}

Signal operator+(Signal sig1, Signal sig2){
		if(sig1.getLength() != sig2.getLength()){
			cout<<"Cannot add signals if different lengths!"<<endl;
			return sig1;
		}

		for(int i = 0; i < sig1.getLength(); i++){
			sig1.addData(sig2.getData(i), i);
		}
		double max = sig1.getMax() > sig2.getMax() ? sig1.getMax() : sig2.getMax();
		sig1.setAvg(sig1.getAvg() + sig2.getAvg());
		sig1.setName("Added");
		sig1.setMax(max);

		return(sig1); }
```