ECE 3220 Lab10
Abstract Base Classes (ABC), Exception Handling

Marshall Lindsay
mblgh6@mail.missouri.edu
14211662

4/26/17

Objective:

The objective for this lab was two-fold: to become familiar with abstract base classes, and to implement exception handling. The first was accomplished by creating a conic ABC that both circles and ellipses are derived from. The latter was through adding try-catch blocks to our existing Lab7 code.

Discussion:

The lab began with the creation of an ABC called BaseConic to derive both a circle and an ellipse from. Each class has its respective constructor to initialize the different data members. Both of the constructors call the BaseConic constructor. The BaseConic class was used to point to the different objects and their methods through dynamic binding. Pure dynamic binding was attempted and created several issues as discussed in the experiment section of this report. Overall the methods used had predictable and valid outputs. However when creating the vertice points for the ellipse, a weird rounding error occurred. This error created numbers extremely close to zero, but never fully rounded down to zero. The outputs are still valid they just add more "junk" to the screen.

Part 2 of this lab was relatively straightforward. I chose several locations to realize try-catch exception handling. Each instance of exception handling will be mentioned briefly here and elaborated on in the experiments section of this report. The first block was implemented on user input when choosing to open the data file via the filename or file number. Throughout this program if the user choses to open a file via the number, the basic filename format "Raw_data_##.txt" (where ## represents the file number replaced with user input) was assumed.

The second instance of try-catch exception handling was on the opening of files to be turned into signals. If the file couldn't be opened the function would throw an exception. Another block is found encasing the entire main loop of the program. This allowed many functions to throw exceptions and those exceptions get caught during runtime. One of such functions was the overloaded addition operator to add two signals. If, when attempting to add the two signals, they weren't the same length, an ExceptionObject was throw. This object contained the length of each of the signals and a method to output the error that occurred. Another function in this main loop that threw exceptions was the scaling function. The way I realized this function back in Lab7 made for a perfect place to implement a rethrow. The try block was based around the user's input to scale the current signal.

Experiments:

Several cases of experimentation was done in part 1 of this lab. Each method was tested in turn and output captured in figures 1 - 10. When attempting to make an object of the ABC a compilation occurred pointing towards the pure virtual functions as errors. Do to the class containing pure virtual functions and therefore being a true ABC, objects of the class cannot be made directly. When using the ABC to point to the different derived classes, if the ABC lacked a virtual version of one of the derived methods, an error was given. Therefore the ABC needed to contain a virtual version of every method used if pure dynamic binding was to be realized. This created more problems when attempting to compile. Every pure virtual function requires an overwriting function. This means that the circle class contains methods that are only used by the ellipse class and vis versa. These functions were left empty and aren't used, but seem unnecessary and probably represent an incorrect use of the ABC. A closer look into more efficient uses of the ABC and dynamic binding will need to be taken.

Part 2 contained a lot of trial and error, especially with the throwing of an object, to insure

proper exception handling. As mentioned in the discussion section, the first try-catch block was found on the user input of whether to open the data file via filename or file number. After conversion to uppercase, if the userInput string did not equal either of the valid inputs, an error message string was throw. This string was caught by a loop in the main function that outputted the error and tried to get another input. On a successful input, the while loop was broken from and the program continued.

The next location for exception handling was on the opening of a file to be used as data for a signal. If the file was not opened, a const char* was thrown by the parametric constructor of the signal class and caught by a catch-all handler in the main program. This const char* contained the filename attempting to be opened. This filename and an error message was displayed. Once the file had been opened and a signal created from the data contents, the main program loop began. A try-catch block was placed around this entire loop so that exceptions could be caught from several functions. The scaling function was one of these functions. Contained in the scaling function was a try-catch block around a function to return user input to be used as a scaling factor. This function, getScalingValue, also contained a try-catch block around the attempt to get user input. If the user failed to enter a number the function would throw a const int equal to 1. If the user entered '0', the function would throw a const int equal to 2. This was caught directly in function where it was thrown and an error message displayed corresponding to the error. The exception was then rethrown back to the scaling function where it was caught and handled. The scaling function would continue this process until a valid user input was obtained. The last location was in the addition overloaded operator to add two signals. If the user attempted to add two signals that were not the same length, an ExceptionObject was throw. This class has two data members, size1 and size2 corresponding to the sizes of the different signals. This object caught by the try-catch block around the main loop and handled accordingly.

Conclusion:

This lab required us to try and experiment with the different characteristics of ABC and exception handling. In order to fully answer all of the questions and also have valid output, a concrete understanding of the different topics was required. This meant by the end of this lab, a much better understanding of abstract base classes, dynamic binding, try blocks, and catch blocks was obtained.

Figures:

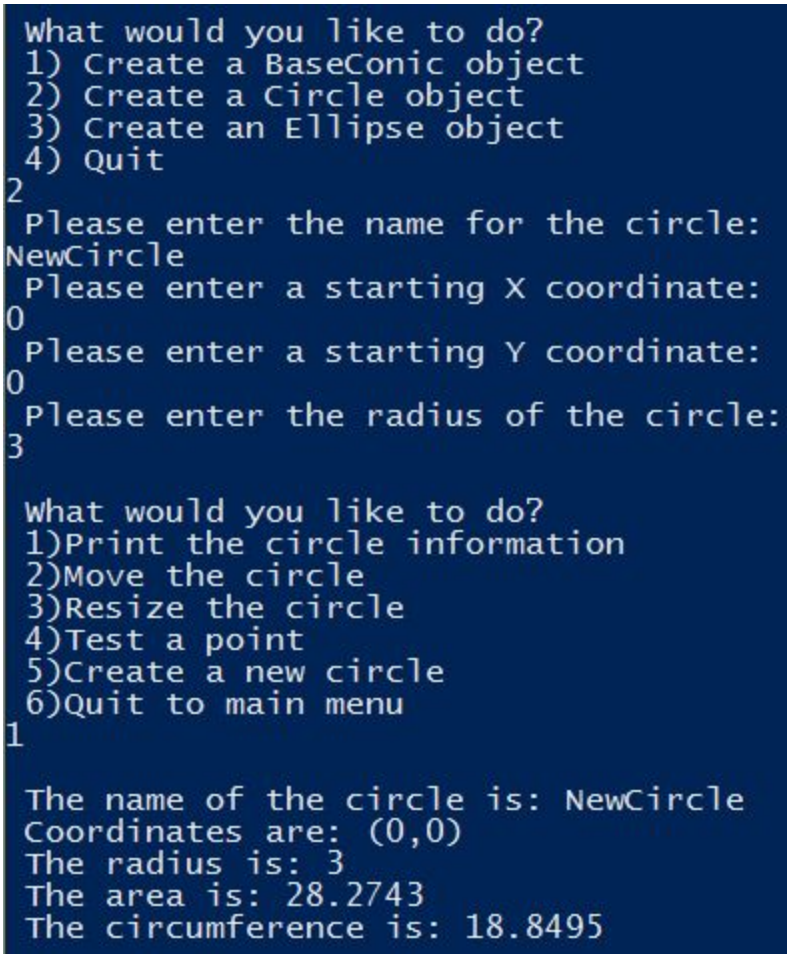

Figure 1: Screenshot showing the creation of a circle and a printout of its information. Initial location was (0,0) and radius = 3.

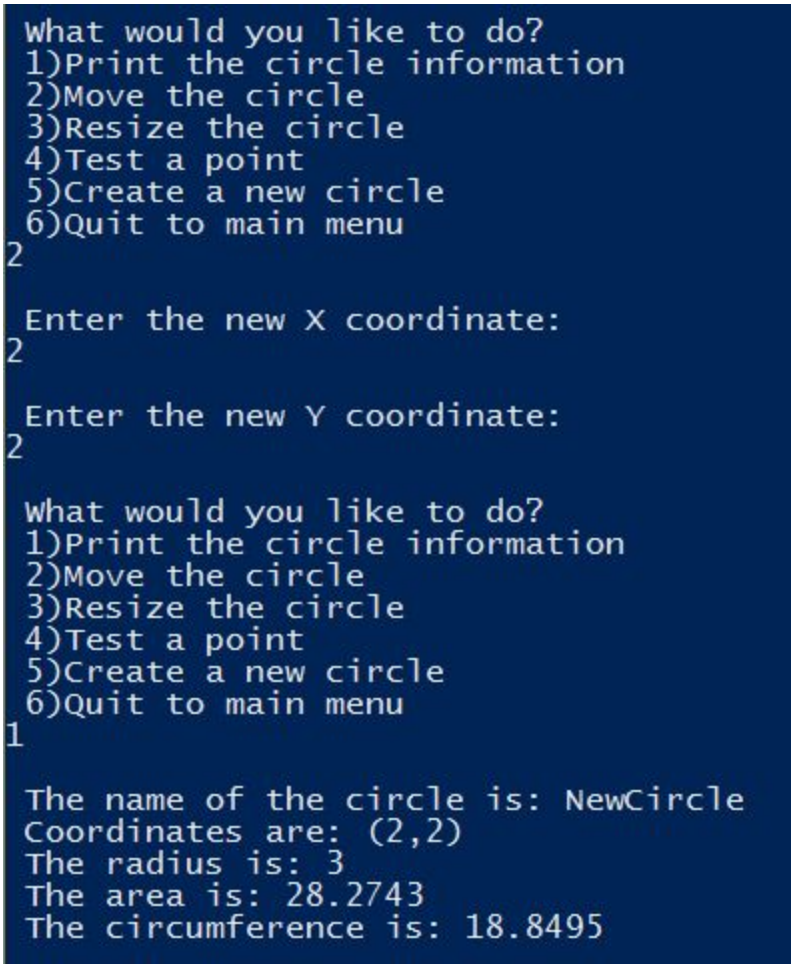

Figure 2: Screenshot showing the move method of the circle. Initial location was (0,0) per Figure 1. Final location was (2,2).

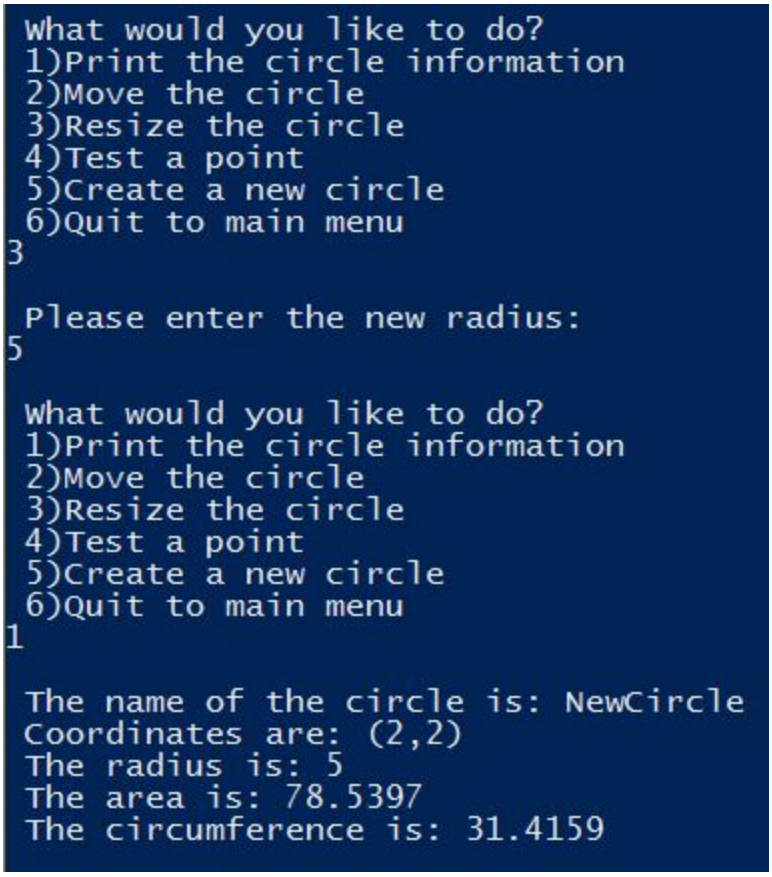

Figure 3: Screenshot showing the use of the resize circle method. Initial radius was 3, final radius was 5. Area and circumference updated accordingly.

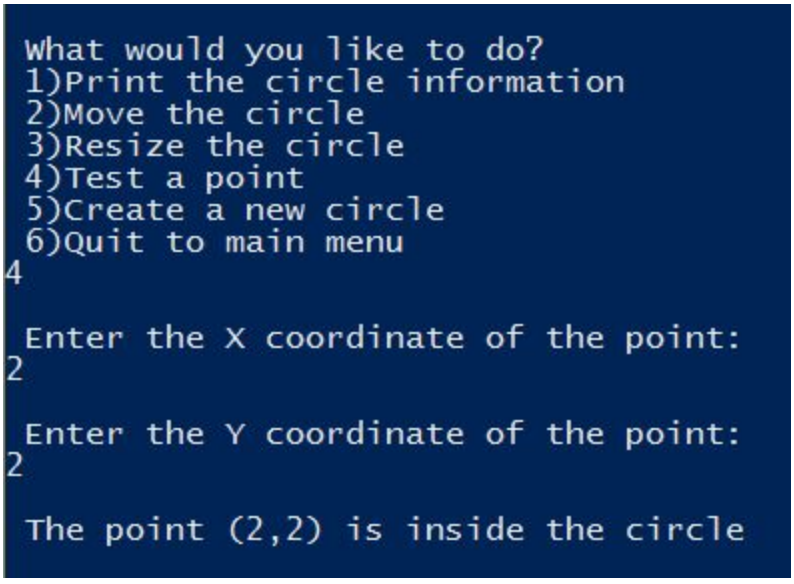

Figure 4: Screenshot showing the testing of point. Circle center was (2,2) and radius = 5. The point entered was (2,2) and was shown to be (obviously) inside the circle.

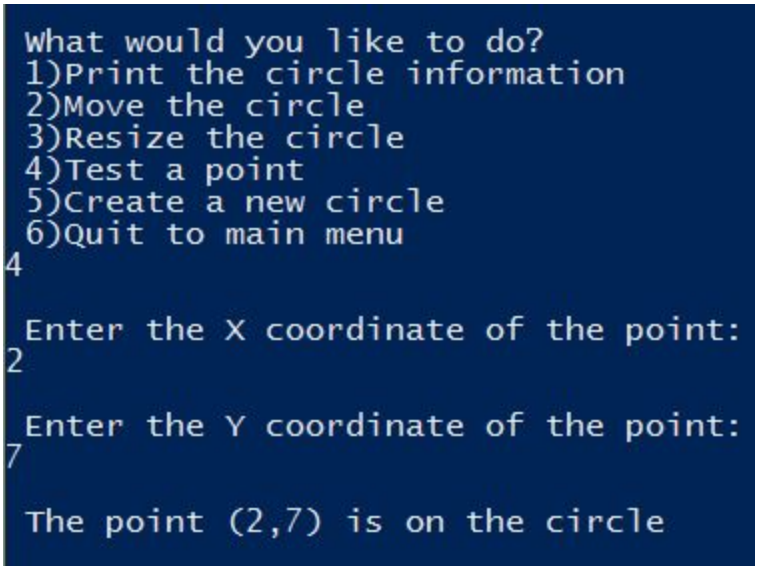

Figure 5: Screenshot showing the testing of a point that was on the circle. Circle characteristics were the same as Figure 4.

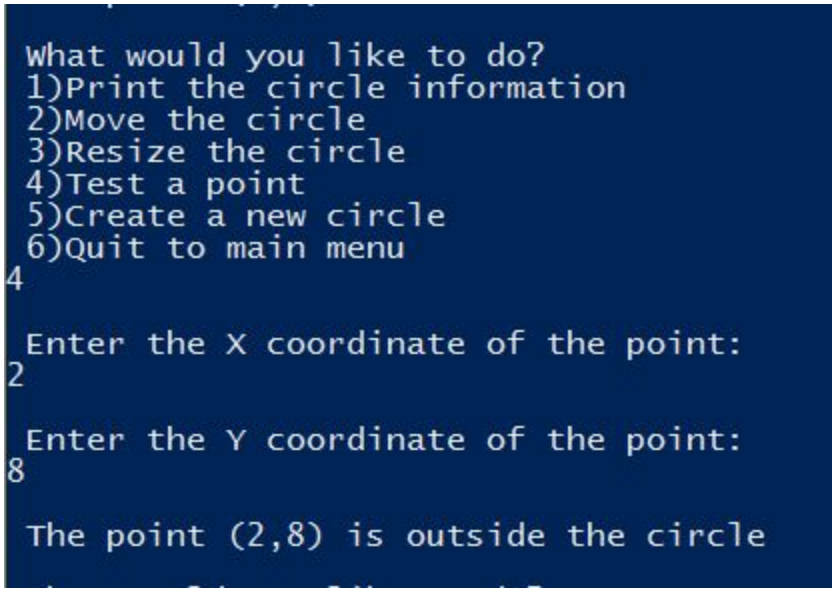

Figure 6: Screenshot showing the testing of a point outside the circle. Circle characteristics were the same as Figures 4 and 5.

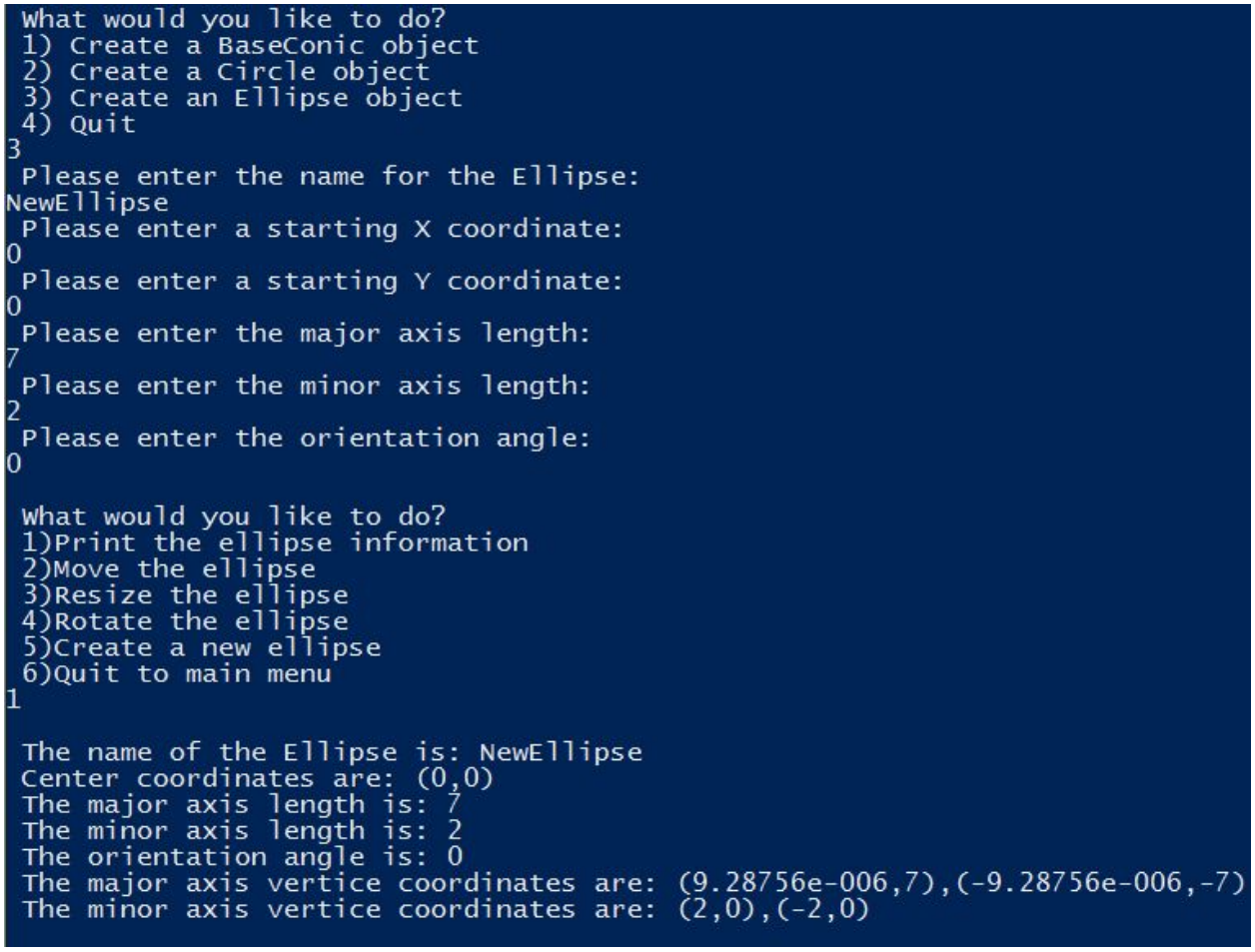

Figure 7: Screenshot showing the creation of an ellipse. Visible here is the rounding error mentioned in the discussion section of this report.

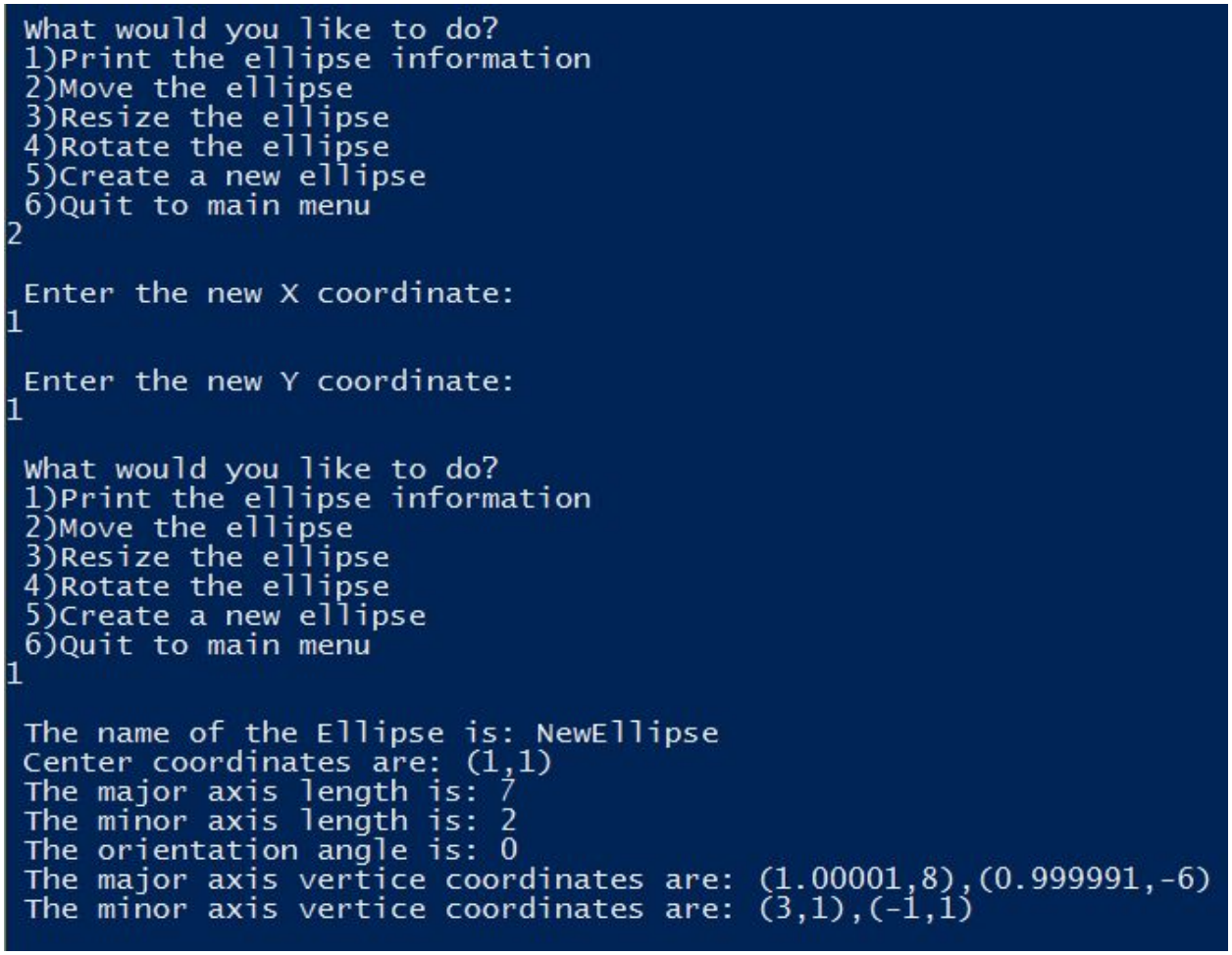

Figure 8: Screenshot showing the execution of the move method for the ellipse. The objects information was printed directly after moving.

```
 What would you like to do?
 1)Print the ellipse information
 2)Move the ellipse
 3)Resize the ellipse
 4)Rotate the ellipse
 5)Create a new ellipse
 6)Quit to main menu
3

 Please enter the new major axis length:
4

 Please enter the new minor axis length:
1

 What would you like to do?
 1)Print the ellipse information
 2)Move the ellipse
 3)Resize the ellipse
 4)Rotate the ellipse
 5)Create a new ellipse
 6)Quit to main menu
1

 The name of the Ellipse is: NewEllipse
 Center coordinates are: (1,1)
 The major axis length is: 4
 The minor axis length is: 1
 The orientation angle is: 0
 The major axis vertice coordinates are: (1.00001,5),(0.999995,-3)
 The minor axis vertice coordinates are: (2,1),(0,1)
```

Figure 9: Screenshot showing the resize method of the ellipse object. Ellipse characteristics should be contrasted with Figure 8.

```
 What would you like to do?
 1)Print the ellipse information
 2)Move the ellipse
 3)Resize the ellipse
 4)Rotate the ellipse
 5)Create a new ellipse
 6)Quit to main menu
4

 Enter the new orientation angle:
90

 What would you like to do?
 1)Print the ellipse information
 2)Move the ellipse
 3)Resize the ellipse
 4)Rotate the ellipse
 5)Create a new ellipse
 6)Quit to main menu
1

 The name of the Ellipse is: NewEllipse
 Center coordinates are: (1,1)
 The major axis length is: 4
 The minor axis length is: 1
 The orientation angle is: 90
 The major axis vertice coordinates are: (-3,1.00001),(5,0.999989)
 The minor axis vertice coordinates are: (1,2),(0.999999,8.80185e-013)
```

Figure 10: Screenshot showing the rotate method of the ellipse object. Ellipse characteristics should be contrasted with Figure 9.

```
.\Lab10.cpp: In function 'BaseConic* createBaseConic()':
.\Lab10.cpp:473:104: error: invalid new-expression of abstract class type 'BaseConic'
 return(new BaseConic(atof(userInput[1].c_str()), atof(userInput[2].c_str()), atof(userInput[3
].c_str())));

         ^
.\Lab10.cpp:17:7: note:   because the following virtual functions are pure within 'BaseConic':
 class BaseConic{
       ^
.\Lab10.cpp:28:16: note:        virtual void BaseConic::rotate(double)
   virtual void rotate(double) = 0;
                ^
.\Lab10.cpp:29:16: note:        virtual void BaseConic::resize()
   virtual void resize(void) = 0;
                ^
.\Lab10.cpp:31:16: note:        virtual void BaseConic::validPoint(double, double)
   virtual void validPoint(double,double) = 0;
                ^
.\Lab10.cpp:32:16: note:        virtual void BaseConic::calcVertices()
   virtual void calcVertices(void) = 0;
                ^
.\Lab10.cpp:33:16: note:        virtual void BaseConic::calcCircumference()
   virtual void calcCircumference(void) = 0;
                ^
.\Lab10.cpp:34:16: note:        virtual void BaseConic::calcArea()
   virtual void calcArea(void) = 0;
                ^
```

Figure 11: Screenshot showing the compilation errors when trying to make an instance of the ABC.

Code:

```
/*
	Marshall Lindsay
	ECE3220
	Lab10 Part 1
	4/25/17
*/
#include <iostream>
#include <string>
#include <math.h>
#include <stdlib.h>
using namespace std;
#define PI 3.14159

/************************************************************
                                        Base Conic Class
************************************************************/
class BaseConic{
	protected:
		string name;	//Name of the object
		double x;	//X coordinate of the center
		double y;	//Y coordinate of the center

	public:
		BaseConic();
		BaseConic(double _x, double _y, string _name);
		virtual ~BaseConic(){};
		virtual void move(double nx, double ny);
		virtual void rotate(double) = 0;
		virtual void resize(void) = 0;
		virtual void printInfo(void);
		virtual void validPoint(double,double) = 0;
		virtual void calcVertices(void) = 0;
		virtual void calcCircumference(void) = 0;
		virtual void calcArea(void) = 0;
};

//Default constructor for BaseConic. Sets X and Y equal to 0 and
//sets the name to "Default"
BaseConic::BaseConic(){
	this->x = 0;
	this->y = 0;
	this->name = "Default";
}

//Parametric constructor for BaseConic. Sets x,y, and name to the
//given values.
BaseConic::BaseConic(double _x, double _y, string _name){
	this -> x = _x;
	this-> y = _y;
	this->name = _name;
}
```

```
//Move method. Sets the x and y coordinates to the given coordinates.
void BaseConic::move(double nx, double ny){
	this->x = nx;
	this->y = ny;
}

//Prints the x and y coordinates of the conic center
void BaseConic::printInfo(void){
	cout<<" This conic's name : "<< this->name
		<<"\n X coordinate for center of : "<<this->x
		<<"\n y coordinate : "<<this->y<<endl;
}

/*************************************************************
				Derived Circle Class from Base Conic
*************************************************************/
class Circle : public BaseConic{
	private:
		double radius;
		double area;
		double circumference;
	public:
		Circle();
		Circle(double _x, double _y, double _radius, string _name);
		~Circle(){};
		void calcCircumference(void);
		void validPoint(double _x, double _y);
		void rotate(double){};
		void resize(void);
		void calcVertices(void){};
		void calcArea(void);
		void printInfo(void);

};

//Default consructor for circle. This should not be used but given just in case.
//Calls the Base Conic default constructor. Sets the radius equal to 1 and calculates
//the area and circumference from there.

Circle::Circle() : BaseConic(){
	this->radius = 1;
	this->calcCircumference();
	this->calcArea();
}

//Parametric constructor. Sets X and Y equal to the given values and calculates the area
//and circumference respectively.

Circle::Circle(double _x, double _y, double _radius, string _name) : BaseConic(_x, _y, _name){
	this->radius = _radius;
	this->calcArea();
	this->calcCircumference();

}
```

```
//Method to set the circumference from the objects radius.
void Circle::calcCircumference(void){
	this->circumference = 2 * PI * this->radius;
}

//Method to set the area from the objects radius.
void Circle::calcArea(void){
	this->area = PI * radius * radius;
}

//Method to resize the object given a new radius.
void Circle::resize(void){
	string userInput;
	cout<<"\n Please enter the new radius: "<<endl;
	getline(cin, userInput);
	this->radius = atof(userInput.c_str());
	this->calcArea();
	this->calcCircumference();
}

//Method to check if a given point is within the circle, outside the circle or on the circle.
//Prints a message corresponding to the result.
void Circle::validPoint(double _x, double _y){
	if(((_x - this->x) * (_x - this->x)) + ((_y- this->y) *(_y - this->y)) == (this->radius * this->radius)){
		cout<<"\n The point ("<<_x<<","<<_y<<") is on the circle"<<endl;
	}else if(((_x - this->x) * (_x - this->x)) + ((_y- this->y) *(_y - this->y)) > (this->radius * this->radius)){
		cout<<"\n The point ("<<_x<<","<<_y<<") is outside the circle"<<endl;
	}else{
		cout<<"\n The point ("<<_x<<","<<_y<<") is inside the circle"<<endl;
	}
}

//Method to print all of the information about the circle.
void Circle::printInfo(void){
	cout<<"\n The name of the circle is: "<<this->name
		<<"\n Coordinates are: ("<<this->x<<","<<this->y<<")"
		<<"\n The radius is: "<<this->radius
		<<"\n The area is: "<<this->area
		<<"\n The circumference is: "<<this->circumference<<endl;
}

/*************************************************************
				Derived Elipse Class from Base Conic
*************************************************************/
class Ellipse : public BaseConic{
	private:
		double a;		//semi-major axis
		double b;			//semi-minor axis
		double angle;	//orientation angle

		double vmajor_x1;
		double vmajor_x2;
		double vmajor_y1;
```

```
		double vmajor_y2;

		double vminor_x1;
		double vminor_x2;
		double vminor_y1;
		double vminor_y2;

		double area;
	public:
		Ellipse();
		Ellipse(double _x, double _y, double _a, double _b, double _angle, string _name);
		~Ellipse(){};
		void move(double nx, double ny);
		void rotate(double nangle);
		void resize(void);
		void printInfo(void);
		void validPoint(double, double){};
		void calcArea(void);
		void calcCircumference(void){};
		void calcVertices(void);
};

//Defaule constructor. Calls the BaseConic default constructor.
//Sets the major axis, minor axis, and angle to zero. Calculates the vertices
//and the area.
Ellipse::Ellipse() : BaseConic(){
	this->a = 1;
	this->b = 1;
	this->angle = 0;
	this->calcArea();
	this->calcVertices();
}

//Parametric constructor. Takes the initial position (x,y), the length of the major
//and minor axis, the angle of orientation and the name of the object. It calls the
//BaseConic parametric constructor with the (x,y) and the name. Calulates the area and
//vertices.
Ellipse::Ellipse(double _x, double _y, double _a, double _b, double _angle, string _name) : BaseConic(_x,_y, _name){
	this->a = _a;
	this->b = _b;
	this->angle = _angle;
	this->calcArea();
	this->calcVertices();
}

//Calculates the area of the Ellipse based off the major and minor axis lengths.
void Ellipse::calcArea(void){
	this->area = PI * a * b;
}

//Calculates the verties of the Ellipse based off the major and minor axis lengths
//and the orientation angle.
void Ellipse::calcVertices(void){
	//Major vertices.
```

```
	this->vmajor_y1 = this->y + (this->a)*(sin((this->angle+90) * PI / 180));
	this->vmajor_x1 = this->x + (this->a)*(cos((this->angle+90) * PI / 180));
	this->vmajor_x2 = this->x - (this->a)*(cos((this->angle+90) * PI / 180));
	this->vmajor_y2 = this->y - (this->a)*(sin((this->angle+90) * PI / 180));

	//Minor vertices.
	this->vminor_x1 = this->x + (this->b)*(cos(this->angle * PI / 180));
	this->vminor_y1 = this->y + (this->b)*(sin(this->angle * PI / 180));
	this->vminor_y2 = this->y - (this->b)*(sin(this->angle * PI / 180));
	this->vminor_x2 = this->x - (this->b)*(cos(this->angle * PI / 180));
}

//Method to rotate the Ellipse. Sets the angle to the new angle and recalculates the
//vertices.
void Ellipse::rotate(double nangle){
		this->angle = nangle;
		this->calcVertices();
}

//Method to move the Ellipse. Sets the center to the new coordinates, calculates the new
//vertices.
void Ellipse::move(double nx, double ny){
	this->x = nx;
	this->y = ny;
	this->calcVertices();
}

//Method to resize the ellipse given new major and minor axis.
void Ellipse::resize(void){
	string userInput;

	cout<<"\n Please enter the new major axis length: "<<endl;
	getline(cin, userInput);
	this->a = atof(userInput.c_str());
	cout<<"\n Please enter the new minor axis length: "<<endl;
	getline(cin, userInput);
	this->b = atof(userInput.c_str());

	this->calcVertices();
}

void Ellipse::printInfo(void){
	cout<<"\n The name of the Ellipse is: "<<this->name
	<<"\n Center coordinates are: ("<<this->x<<","<<this->y<<")"
	<<"\n The major axis length is: "<<this->a
	<<"\n The minor axis length is: "<<this->b
	<<"\n The orientation angle is: "<<this->angle
	<<"\n The major axis vertice coordinates are: ("<<this->vmajor_x1<<","<<this->vmajor_y1<<"),"
	<<"("<<this->vmajor_x2<<","<<this->vmajor_y2<<")"
	<<"\n The minor axis vertice coordinates are: ("<<this->vminor_x1<<","<<this->vminor_y1<<"),"
	<<"("<<this->vminor_x2<<","<<this->vminor_y2<<")"<<endl;

}
```

```
void optionsMenu(void);

void baseConic(BaseConic*);
BaseConic* createBaseConic(void);

void ellipse(BaseConic*);
BaseConic* createEllipse(void);
void ellipseOptionsMenu(void);

void circle(BaseConic*);
BaseConic* createCircle(void);
void circleOptionsMenu(void);

int main(void){
	BaseConic* basePtr = NULL;

	string userInput;
	while(1){
		optionsMenu();
		getline(cin,userInput);
		if(userInput == "1"){
			baseConic(basePtr);
		}else if(userInput == "2"){
			circle(basePtr);
		}else if(userInput == "3"){
			ellipse(basePtr);
		}else if(userInput == "4"){
			return(0);
		}else{
			cout<<"\n Invalid Input!"<<endl;
		}

	}

	return(0);
}

void optionsMenu(void){
	cout<<"\n What would you like to do?"<<endl;
	cout<<" 1) Create a BaseConic object\n"
		<<" 2) Create a Circle object\n"
		<<" 3) Create an Ellipse object\n"
		<<" 4) Quit"<<endl;
}

void circle(BaseConic* circlePtr){
	string userInput[2];

	circlePtr = createCircle();

	while(1){
		circleOptionsMenu();
		getline(cin, userInput[0]);
		if(userInput[0] == "1"){
```

```
                circlePtr->printInfo();
            }else if(userInput[0] == "2"){
                cout<<"\n Enter the new X coordinate:"<<endl;
                getline(cin, userInput[0]);
                cout<<"\n Enter the new Y coordinate:"<<endl;
                getline(cin, userInput[1]);

                circlePtr->move(atof(userInput[0].c_str()), atof(userInput[1].c_str()));

            }else if(userInput[0] == "3"){
                circlePtr->resize();
            }else if(userInput[0] == "4"){
                cout<<"\n Enter the X coordinate of the point:"<<endl;
                getline(cin, userInput[0]);
                cout<<"\n Enter the Y coordinate of the point:"<<endl;
                getline(cin, userInput[1]);
                circlePtr->validPoint(atof(userInput[0].c_str()), atof(userInput[1].c_str()));

            }else if(userInput[0] == "5"){
                delete circlePtr;
                circlePtr = createCircle();

            }else if(userInput[0] == "6"){
                delete circlePtr;
                break;
            }else{
                cout<<"\n Invalid Input!"<<endl;
            }
        }

}

BaseConic* createCircle(void){
        string userInput[4];
        cout<<" Please enter the name for the circle: "<<endl;
        getline(cin, userInput[0]);

        cout<<" Please enter a starting X coordinate: "<<endl;
        getline(cin, userInput[1]);

        cout<<" Please enter a starting Y coordinate: "<<endl;
        getline(cin, userInput[2]);

        cout<<" Please enter the radius of the circle: "<<endl;
        getline(cin, userInput[3]);

        return(new Circle(atof(userInput[1].c_str()), atof(userInput[2].c_str()), atof(userInput[3].c_str()), userInput[0]));

}

void circleOptionsMenu(void){
        cout<<"\n What would you like to do?"
            <<"\n 1)Print the circle information"
            <<"\n 2)Move the circle"
```

```
                <<"\n 3)Resize the circle"
                <<"\n 4)Test a point"
                <<"\n 5)Create a new circle"
                <<"\n 6)Quit to main menu"<<endl;
}

void ellipse(BaseConic* ellipsePtr){
        string userInput[2];

        ellipsePtr = createEllipse();

        while(1){
                ellipseOptionsMenu();
                getline(cin, userInput[0]);
                if(userInput[0] == "1"){
                        ellipsePtr->printInfo();
                }else if(userInput[0] == "2"){
                        cout<<"\n Enter the new X coordinate:"<<endl;
                        getline(cin, userInput[0]);
                        cout<<"\n Enter the new Y coordinate:"<<endl;
                        getline(cin, userInput[1]);

                        ellipsePtr->move(atof(userInput[0].c_str()), atof(userInput[1].c_str()));

                }else if(userInput[0] == "3"){
                        ellipsePtr->resize();
                }else if(userInput[0] == "4"){
                        cout<<"\n Enter the new orientation angle: "<<endl;
                        getline(cin, userInput[0]);
                        ellipsePtr->rotate(atof(userInput[0].c_str()));
                }else if(userInput[0] == "5"){
                        delete ellipsePtr;
                        ellipsePtr = createEllipse();

                }else if(userInput[0] == "6"){
                        delete ellipsePtr;
                        break;
                }else{
                        cout<<"\n Invalid Input!"<<endl;
                }
        }
}

BaseConic* createEllipse(void){
        string userInput[6];
        cout<<" Please enter the name for the Ellipse: "<<endl;
        getline(cin, userInput[0]);

        cout<<" Please enter a starting X coordinate: "<<endl;
        getline(cin, userInput[1]);

        cout<<" Please enter a starting Y coordinate: "<<endl;
        getline(cin, userInput[2]);
```

```
	cout<<" Please enter the major axis length: "<<endl;
	getline(cin, userInput[3]);

	cout<<" Please enter the minor axis length: "<<endl;
	getline(cin, userInput[4]);

	cout<<" Please enter the orientation angle: "<<endl;
	getline(cin, userInput[5]);

	return(new Ellipse(atof(userInput[1].c_str()), atof(userInput[2].c_str()),
				atof(userInput[3].c_str()), atof(userInput[4].c_str()), atof(userInput[5].c_str()), userInput[0]));
}

void ellipseOptionsMenu(void){
	cout<<"\n What would you like to do?"
		<<"\n 1)Print the ellipse information"
		<<"\n 2)Move the ellipse"
		<<"\n 3)Resize the ellipse"
		<<"\n 4)Rotate the ellipse"
		<<"\n 5)Create a new ellipse"
		<<"\n 6)Quit to main menu"<<endl;
}

void baseConic(BaseConic* basePtr){

	string userInput[2];
	cout<<"Cannot create an object that is an ABC"<<endl;
	//basePtr = createBaseConic();

}
/*
BaseConic* createBaseConic(void){
	string userInput[3];
	cout<<" Please enter the name for the Ellipse: "<<endl;
	getline(cin, userInput[0]);

	cout<<" Please enter a starting X coordinate: "<<endl;
	getline(cin, userInput[1]);

	cout<<" Please enter a starting Y coordinate: "<<endl;
	getline(cin, userInput[2]);

return(new BaseConic(atof(userInput[1].c_str()), atof(userInput[2].c_str()), atof(userInput[3].c_str())));
}
*/

/*****************************************************************************************************/

/*
	Marshall Lindsay
	Lab10
	4/25/17
*/
```

```
#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)throw(const char*);
		~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)throw(const char*){
	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()){
		throw(fileName);
	}

	//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;
}

class ExceptionObject{
	private:
		int size1;
		int size2;
	public:
		ExceptionObject(int a, int b) : size1(a), size2(b){};
		void msg(void);

};

void ExceptionObject::msg(void){
	cout<<"\n Could not add the two signals, they are not the same size!\n"
		<<"\n Signal1 size: "<<this->size1
		<<"\n Signal2 size: "<<this->size2<<endl;
}

Signal operator+(Signal, Signal) throw(ExceptionObject);
void optionMenu();
double scaling();
double offsetting();
Signal addSignals();
string fileSave();
string getUserInput1(void) throw(string);
double getScalingValue(void) throw(const int);

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){

		while(1){
			try{
				userInput = getUserInput1();
				break;
			}
			catch(string message){
				cout<<message;
```

```
		}
	}

	if(userInput == "F"){
		while(1){
			try{
				cout<<"Please enter the file name to be opened:"<<endl;
				getline(cin, fileName);
				dataSignal = new Signal(fileNamePtr);			//If file does not exist, will throw an exception.
				break;
			}
			catch(...){
	//Catch will catch the bad file exception.
				cout<<"\n "<<fileName<<" is not a valid file!"<<endl;
				cin.clear();
				fflush(stdin);
			}
		}
	}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){
	try{
			//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;
                        }
                }
                catch(ExceptionObject eo){
                        eo.msg();
                }
                catch(...){
                        cout<<"\n Catch all block from optionsMenu"<<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;
	while(1){
		try{
			value = getScalingValue();
			return(value);
		}
		catch(const int y){
			cout<<"\n Rethrow caught in scaling. Supplied by getScalingValue"<<endl;
			if(y == 1){
				cin.clear();
				fflush(stdin);
			}
		}
		catch(...){
			cout<<"\n Rethrow caught in scaling(catch all). Supplied by getScalingValue"<<endl;
		}
	}
}

double getScalingValue()throw(const int){
	double userInput;
	try{
		cout<<"\n Enter a value to scale by: "<<endl;
		cin>>userInput;
		if(cin.fail()){
			throw(1);
		}else if(userInput == 0){
			throw(2);
		}
		return(userInput);
	}
	catch(const int x){
		if(x == 1){
			cout<<"\n Scaling value supplied is not a number!"<<endl;
			throw;
		}else if(x == 2){
			cout<<"\n Cannot scale by zero!"<<endl;
			throw;
		}
	}
	catch(...){
		cout<<"\n Not sure how you got here. Catch all getScalingValue()"<<endl;
		throw;
	}

}
```

```
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) throw(ExceptionObject){
	if(sig1.getLength() != sig2.getLength()){
		throw(ExceptionObject(sig1.getLength(), sig2.getLength()));
	}

	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);

}

string getUserInput1(void) throw(string){
	string userInput;
	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);
	}

	if(userInput != "F" && userInput != "N")
		throw((string)("\n Invalid Input!"));
	else
		return(userInput);}
```