```cpp
/*
    Marshall Lindsay
    Max Houck
    Formula SAE Tire Temperature Visualization
    ECE 3220 Final Project
    tires.cpp
*/


#include <iostream>
#include <vector>
#include <fstream>
#include <string>
#include "tires.h"
using namespace std;

/*****************************************************************************
                          TIRE CLASS METHODS
*****************************************************************************/
/*****************************************************************************
    Tire class parametric constuctor. Takes the location of the new tire as
the tireLocation enum as well as the name of the new tire and sets the
corresponding values in the object.

*****************************************************************************/
Tire::Tire(tireLocation tireLoc, string newName){
    this->location = tireLoc;
    this->name = newName;
}

/*****************************************************************************
    Tire class addSensor method. Takes a Sensor object as an arguement
and adds it to the sensorArray vector.

*****************************************************************************/
void Tire::addSensor(Sensor newSensor){
    this->sensorArray.push_back(newSensor);
}


/*****************************************************************************
    Tire class removeSensor method. Accepts the location of the sensor
as a sensorLocation enum and removes the selected sensor from the
sensorArray vector. It also checks to make sure the senssorArray vector
is not empty. If the vector is empty, it will throw an const int equal to
1.

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

void Tire::removeSensor(sensorLocation sensorLoc) throw(const int){
    //If the sensor array is empty, throw the number 1
    if(this->sensorArray.size() <= 0){
        throw(1);
    }

    for(int i = 0; i < this->sensorArray.size(); i++){
        if(this->sensorArray[i].location == sensorLoc){
            this->sensorArray.erase(this->sensorArray.begin()+i);
            break;
        }
    }

}

/*****************************************************************************
    Tire class printInfo method. Prints all of the information about the
tire to the screen.
*****************************************************************************/
void Tire::printInfo(void){
```

```
70      cout<<"\n Name: "<<this->name
71          <<"\n Location: "<< this->location<<endl;
72
73
74      cout<<" Sensor vector contents: "<<endl;
75      if(this->sensorArray.size() <= 0){
76          cout<<"  Sensor array is empty!"<<endl;
77      }else{
78          for(int i = 0; i < this->sensorArray.size(); i++){
79              sensorArray[i].printInfo();
80          }
81      }
82  }
83
84  /*************************************************************************
85                          SENSOR CLASS METHODS
86  *************************************************************************/
87  /*************************************************************************
88      Sensor class parametric constructor. Takes the name, the location as a
89  sensorLoaction enum, and the address as arguements. Sets the corresponding
90  fields to the given values.
91
92  *************************************************************************/
93  Sensor::Sensor(string newName, sensorLocation sensorLoc, int sensorAddress){
94      this->name = newName;
95      this->location = sensorLoc;
96      this->address = sensorAddress;
97      this->ambTemperature = 0;
98  }
99
100 /*************************************************************************
101     Sensor class addTemperature method. Takes a temperature as an integer
102 and adds it to the objTemperature vector.
103 *************************************************************************/
104 void Sensor::addTemperature(int temperature){
105     this->objTemperature.push_back(temperature);
106 }
107
108 /*************************************************************************
109     Sensor class getTemperature method. Returns the most recent
110 temperature from the objTemperature vector. Checks to make sure the
111 objTemperature vector is not empty. If the vector is empty, returns a
112 const int equal to 2.
113 *************************************************************************/
114 int Sensor::getTemperature(void) throw(const int){
115     if(this->objTemperature.size() <= 0){
116         throw(2);
117     }
118     return(this->objTemperature.back());
119 }
120
121 /*************************************************************************
122     Sensor class printEverything method. Used for debugging, this method
123 does exactly what the name implies. It will print everything contained in
124 the class to the screen.
125 *************************************************************************/
126 void Sensor::printInfo(void){
127     cout<<"\n  "<<this->name<<" information: "
128         <<"\n   Sensor location: "<<this->location
129         <<"\n   Ambient temperature: "<<this->ambTemperature
130         <<"\n   Address : "<<this->address<<endl;
131
132     cout<<"   Temperature vector contents: "<<endl;
133     if(this->objTemperature.size() <= 0){
134         cout<<"   The vector is empty!"<<endl;
135     }else{
136         for(int i = 0; i < this->objTemperature.size(); i++){
137             cout<<"    "<<this->objTemperature[i];
138             if(i % 3 == 0)
```

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

/*****************************************************************************
                        CAR CLASS METHODS
*****************************************************************************/
/*****************************************************************************
    Car class parametric constructor. Takes the name of the car as a string
and sets the corresponding field.
*****************************************************************************/
Car::Car(string newName){
    this->name = newName;
}

/*****************************************************************************
    Car class printInfo method. Prints all of the information about the car
to the screen.
*****************************************************************************/
void Car::printInfo(void){
        cout<<"\n"<<this->name<<endl;
        cout<<"Tire vector contents: "<<endl;
        if(this->tireArray.size() <= 0){
            cout<<"Tire array is empty!"<<endl;
        }else{
            for(int i = 0; i < this->tireArray.size(); i++){
                    cout<<"\n Tire "<<i<<endl;
                    tireArray[i].printInfo();
            }
        }
        cout<<"-----------------------------------"<<endl;
}

/*****************************************************************************
    Car class getTemperature method. Takes the tire location,
sensor location (as tireLocation and sensorLocation respectively), and the
index of the temperature value. Returns the temperature at that index as
an integer.
*****************************************************************************/
int Car::getTemperature(tireLocation tireLoc, sensorLocation sensorLoc, int index){
    return(this->tireArray[tireLoc-1].sensorArray[sensorLoc-1].objTemperature[index]);
}
/*****************************************************************************
    Car class numDataPoints method. Returns the number of data points in
the temperature vectors as an integer.
*****************************************************************************/
int Car::numDataPoints(void){
    return(this->tireArray[0].sensorArray[0].objTemperature.size());
}
/*****************************************************************************
    Car class addTire method. Takes a Tire object as an arguement and
adds it to the tire vector.
*****************************************************************************/
void Car::addTire(Tire newTire){
    this->tireArray.push_back(newTire);
}




//Parametric constructor. This will be the only way a badVector is created.
badVector::badVector(vector<int> vect, char tireLoc, char vectorLoc){
    this->size = vect.size();
    this->tireLocation = tireLoc;
    this->vectorLocation = vectorLoc;
}

```

```
208  //Default destructor.
209  badVector::~badVector(void){
210      //Nothing is needed here.
211  }
212
213  //Prints the message corresponding to the error.
214  void badVector::badVectorMsg_InvalidSize(void){
215          string tireName;
216          string vectorLocation;
217          switch(this->tireLocation){
218              case 1:
219                  tireName = "frontLeft";
220                  break;
221              case 2:
222                  tireName = "frontRight";
223                  break;
224              case 3:
225                  tireName = "rearLeft";
226                  break;
227              case 4:
228                  tireName = "rearRight";
229                  break;
230              default:
231                  cout<<"Error in badVectorMsg_InvalidSize"<<endl;
232                  exit(2);
233                  break;
234
235          }//End switch
236          switch(this->vectorLocation){
237              case 'o':
238                  vectorLocation = "outer";
239                  break;
240              case 'm':
241                  vectorLocation = "middle";
242              case 'i':
243                  vectorLocation = "inner";
244              default:
245                  cout<<"Error in badVectorMsd_InvalidSize vectorLocation switch"<<endl;
246                  exit(3);
247                  break;
248
249          }//End switch
250          cout<<"Could not access "<< tireName<< " "<<vectorLocation<< " vector. This
             vectors size is "<< this->size<<endl;
251  }
```