```
********************
**                **
**   PROBLEM #1   **
**                **
********************

** CODE **

/*
 *  Project:    HW #2, Problem #1
 *  Author:     Matthew Springer
 *  Date:       February 4, 2017
 *  Purpose:    Write an object-oriented program that produces an array
of
 *              CarRecords from an input file and performs operations
on the
 *              array
 */

// Libraries
#include <iostream>
#include <fstream>
#include <string>
#include <cmath>

using namespace std;

/*
 *  Class Name:  Car
 *
 *  PRIVATE MEMBERS:
 *  @param  make  the make of the car
 *  @param  model the model of the car
 *  @param  year  the year of the car
 *  @param  color the color of the car
 */
class Car {

private:
  string make;
  string model;
  int year;
  string color;

public:
  Car() {}

  ~Car() {}

  void setFields(string mk, string md, int yr, string cl) {
    make = mk;
    model = md;
    year = yr;
    color = cl;
  }
```

```
string getMake() {
    return make;
  }

  string getModel() {
    return model;
  }

  int getYear() {
    return year;
  }

  string getColor() {
    return color;
  }
};

/*
 *  Class Name:  CarRecords
 *
 *  PRIVATE MEMBERS:
 *  @param  arraySize size of the array set to the number of records
read from
 *                    the file
 *  @param  infile    input file stream used to read information from a
file
 *  @param  filename  filename for infile to open;
 *  @param  lines     number of records in file "filename"
 *  @param  cars      a pointer object (to a dynamically allocated
array of
 *                    objects)
 */
class CarRecords {

private:
  int arraySize;
  ifstream infile;
  string filename;
  int lines;
  Car *cars;
```

```
/*
   *  Function: insertArray
   *
   *  Purpose:  populate the array of Cars from the file
"CarRecords.txt"
   */
  void insertArray() {
    string mk;
    string md;
    string cl;
    string yr;
    int yrInt;
    infile.open(filename);
    for (int i = 0; i < arraySize; i++) {
      if(getline(infile, mk, ',') &&
         getline(infile, md, ',') &&
         getline(infile, yr, ',') &&
         getline(infile, cl)){
        yrInt = stoi(yr);
        cars[i].setFields(mk, md, yrInt, cl);
      }
    }
    infile.close();
  }

public:

  CarRecords(int size) {
    filename = "CarRecords.txt";
    lines = 10;
    // constrain 0 <= arraySize <= lines in file
    arraySize = max(0, min(size, lines));
    cars = new Car[arraySize];
    insertArray();
  }

  ~CarRecords() {
    delete [] cars;
  }

  /*
   *  Function: printCarRecords
   *
   *  Purpose:  print the array of Cars to the console
   */
  void printCarRecords() {
    cout << endl << "PRINTING " << arraySize << " RECORDS!" << endl;
    cout << "---------------------" << endl;
    string comma = ", ";
    for (int i = 0; i < arraySize; i++) {
      cout << cars[i].getMake() << comma << cars[i].getModel() << comma
<<
        cars[i].getYear() << comma << cars[i].getColor() << endl;
    }
    cout << endl;
  }
  /*
   *  Function: sortCarRecordsByMake
   *
   *  Purpose:  sort the array of Cars alphabetically by make
   */
  void sortCarRecordsByMake() {
    cout << "SORTING RECORDS BY MAKE...";
    int lowestIndex;
    Car * placeholder = new Car();
    for (int i = 0; i < arraySize - 1; i++) {
      lowestIndex = i;
      for (int j = i+1; j < arraySize; j++) {
        if (cars[j].getMake().compare(cars[lowestIndex].getMake()) < 0
) {
          lowestIndex = j;
        }
      }
      if (lowestIndex > i) {
        placeholder->setFields(cars[i].getMake(),
                               cars[i].getModel(),
                               cars[i].getYear(),
                               cars[i].getColor());

        cars[i].setFields(cars[lowestIndex].getMake(),
                          cars[lowestIndex].getModel(),
                          cars[lowestIndex].getYear(),
                          cars[lowestIndex].getColor());

        cars[lowestIndex].setFields(placeholder->getMake(),
                                    placeholder->getModel(),
                                    placeholder->getYear(),
                                    placeholder->getColor());
      }
    }
    delete placeholder;
    cout << "DONE" << endl;
  }
```

```
/*
   *  Function: sortCarRecordsByYear
   *
   *  Purpose:  sort the array of Cars by year from lowest->highest
   */
  void sortCarRecordsByYear() {
    cout << "SORTING RECORDS BY YEAR...";
    int lowestIndex;
    Car * placeholder = new Car();
    for (int i = 0; i < arraySize - 1; i++) {
      lowestIndex = i;
      for (int j = i+1; j < arraySize; j++) {
        if (cars[j].getYear() < cars[lowestIndex].getYear()) {
          lowestIndex = j;
        }
      }
      if (lowestIndex > i) {
        placeholder->setFields(cars[i].getMake(),
                               cars[i].getModel(),
                               cars[i].getYear(),
                               cars[i].getColor());

        cars[i].setFields(cars[lowestIndex].getMake(),
                          cars[lowestIndex].getModel(),
                          cars[lowestIndex].getYear(),
                          cars[lowestIndex].getColor());

        cars[lowestIndex].setFields(placeholder->getMake(),
                                    placeholder->getModel(),
                                    placeholder->getYear(),
                                    placeholder->getColor());
      }
    }
    delete placeholder;
    cout << "DONE" << endl;
  }
```

```
/*
   *  Function: printDuplicates
   *
   *  Purpose:  print any duplicate Cars in the array
   */
  void printDuplicates() {
    cout << "CHECKING FOR DUPLICATES...";
    bool hasDupes = false;
    // initialize an array of equal length (in case all records are
duplicates)
    bool * duplicates = new bool[arraySize];
    for (int i = 0; i < arraySize - 1; i++) {
      for (int j = i+1; j < arraySize; j++) {
        if (cars[i].getMake() == cars[j].getMake() &&
            cars[i].getModel() == cars[j].getModel() &&
            cars[i].getYear() == cars[j].getYear() &&
            cars[i].getColor() == cars[j].getColor()) {
          duplicates[i] = true;
          duplicates[j] = true;
          hasDupes = true;
        }
      }
    }
    cout << "DONE" << endl << endl;
    if (hasDupes) {
      string comma = ", ";
      for (int i = 0; i < arraySize; i++) {
        if (duplicates[i]) {
          cout << cars[i].getMake() << comma << cars[i].getModel() <<
comma <<
            cars[i].getYear() << comma << cars[i].getColor() << endl;
        }
      }
      cout << endl;
    }
    else cout << "NO DUPLICATE ENTRIES" << endl << endl;
  }

};
```

```
int main() {
  int numRecs;
  cout << "Number or Records to read? " ;
  cin >> numRecs;
  CarRecords *cr = new CarRecords(numRecs);
  // Print car records
  cr->printCarRecords();
  // Sort by Year
  cr->sortCarRecordsByYear();
  // Print car records
  cr->printCarRecords();
  // Sort by Make
  cr->sortCarRecordsByMake();
  // Print car records
  cr->printCarRecords();
  // Check for Duplicates
  cr->printDuplicates();
  delete cr;
} // end main
```

```
** CONSOLE OUTPUT **

Matts-MacBook-Pro:assign1 mspringer$ g++ CarRecords.cpp -o CarRecords
Matts-MacBook-Pro:assign1 mspringer$ ./CarRecords
Number or Records to read? 15

PRINTING 10 RECORDS!
---------------------
Subaru, Outback, 2016, green
Toyota, Corolla, 2006, white
Dodge, Neon, 1993, pink
Ford, Fusion, 2013, yellow
Honda, Fit, 2015, blue
Ford, Expedition, 2009, silver
Toyota, Corolla, 2006, white
Ford, Fusion, 2013, yellow
Jeep, Cherokee, 1999, red
Mazda, Protoge, 1996, gold

SORTING RECORDS BY YEAR...DONE

PRINTING 10 RECORDS!
---------------------
Dodge, Neon, 1993, pink
Mazda, Protoge, 1996, gold
Jeep, Cherokee, 1999, red
Toyota, Corolla, 2006, white
Toyota, Corolla, 2006, white
Ford, Expedition, 2009, silver
Ford, Fusion, 2013, yellow
Ford, Fusion, 2013, yellow
Honda, Fit, 2015, blue
Subaru, Outback, 2016, green

SORTING RECORDS BY MAKE...DONE
```

```
PRINTING 10 RECORDS!
---------------------
Dodge, Neon, 1993, pink
Ford, Expedition, 2009, silver
Ford, Fusion, 2013, yellow
Ford, Fusion, 2013, yellow
Honda, Fit, 2015, blue
Jeep, Cherokee, 1999, red
Mazda, Protoge, 1996, gold
Subaru, Outback, 2016, green
Toyota, Corolla, 2006, white
Toyota, Corolla, 2006, white

CHECKING FOR DUPLICATES...DONE

Ford, Fusion, 2013, yellow
Ford, Fusion, 2013, yellow
Toyota, Corolla, 2006, white
Toyota, Corolla, 2006, white

Matts-MacBook-Pro:assign1 mspringer$ ./CarRecords
Number or Records to read? 7

PRINTING 7 RECORDS!
---------------------
Subaru, Outback, 2016, green
Toyota, Corolla, 2006, white
Dodge, Neon, 1993, pink
Ford, Fusion, 2013, yellow
Honda, Fit, 2015, blue
Ford, Expedition, 2009, silver
Toyota, Corolla, 2006, white

SORTING RECORDS BY YEAR...DONE

PRINTING 7 RECORDS!
---------------------
Dodge, Neon, 1993, pink
Toyota, Corolla, 2006, white
Toyota, Corolla, 2006, white
Ford, Expedition, 2009, silver
Ford, Fusion, 2013, yellow
Honda, Fit, 2015, blue
Subaru, Outback, 2016, green

SORTING RECORDS BY MAKE...DONE

PRINTING 7 RECORDS!
---------------------
Dodge, Neon, 1993, pink
Ford, Expedition, 2009, silver
Ford, Fusion, 2013, yellow
Honda, Fit, 2015, blue
Subaru, Outback, 2016, green
Toyota, Corolla, 2006, white
Toyota, Corolla, 2006, white
```

```
CHECKING FOR DUPLICATES...DONE

Toyota, Corolla, 2006, white
Toyota, Corolla, 2006, white
```

```
********************
**                **
**   PROBLEM #2   **
**                **
********************

** CODE **

/*
 *  @file   Furniture.h
 *  @author Matthew Springer
 *  @date   February 7, 2016
 */

#ifndef FURNITURE_H
#define FURNITURE_H

#include <string>

/*
 *  Class Name:  Furniture
 *
 *  PRIVATE MEMBERS:
 *  @param  width   the width of the furniture
 *  @param  height  the height of the furniture
 *  @param  depth   the depth of the furniture
 *  @param  name    the unique name of the furniture
 */
class Furniture {

private:

  float width;
  float height;
  float depth;
  std::string name;

public:

  // constructor
  Furniture(std::string nm);

  // destructor
  ~Furniture();

  /*
   *  Function: readDimensions
   *
   *  Purpose:  initialize width, height, and depth from user input
   */
  void readDimensions();
/*
   *  Function: print
   *
   *  Purpose:  print information about the furniture to stdout
   */
  virtual void print();
};

#endif
```

```
/*
 *  @file   Bed.h
 *  @author Matthew Springer
 *  @date   February 7, 2016
 */

#ifndef BED_H
#define BED_H

#include <string>

#include "Furniture.h"

/*
 *  Class Name:  Bed
 *
 *  INHERITED MEMBERS FROM FURNITURE:
 *  @param  width   the width of the furniture
 *  @param  height  the height of the furniture
 *  @param  depth   the depth of the furniture
 *  @param  name    the unique name of the furniture
 *
 *  PRIVATE MEMBERS:
 *  @param  bedSize the size of the bed ("Twin", "Full", "Queen", or
"King")
 */
class Bed: public Furniture {

private:

  std::string bedSize;

public:

  // constructor
  Bed(std::string nm, std::string sz);

  // destructor
  ~Bed();

  /*
   *  Function: print
   *
   *  Purpose:  print information about the bed to stdout
   */
  void print();
};

#endif
```

```
/*
 *  @file   Table.h
 *  @author Matthew Springer
 *  @date   February 7, 2016
 */

#ifndef TABLE_H
#define TABLE_H

#include <string>

#include "Furniture.h"

/*
 *  Class Name:  Table
 *
 *  INHERITED MEMBERS FROM FURNITURE:
 *  @param  width   the width of the furniture
 *  @param  height  the height of the furniture
 *  @param  depth   the depth of the furniture
 *  @param  name    the unique name of the furniture
 *
 *  PRIVATE MEMBERS:
 *  @param  bedSize the type of wood ("Pine" or "Oak")
 */
class Table: public Furniture {

private:

  std::string wood;

public:

  // constructor
  Table(std::string nm, std::string wd);

  // destructor
  ~Table();

  /*
   *  Function: print
   *
   *  Purpose:  print information about the table to stdout
   */
  void print();
};

#endif
```

```
/*
 *  @file   Furniture.cpp
 *  @author Matthew Springer
 *  @date   February 7, 2016
 */

#include <iostream>
#include <string>

#include "Furniture.h"

using namespace std;

// constructor
Furniture::Furniture(string nm) {
  name = nm;
}

// destructor
Furniture::~Furniture() {}

/*
 *  Function: readDimensions
 *
 *  Purpose:  initialize width, height, and depth from user input
 */
void Furniture::readDimensions() {
  float w, h, d;
  bool valid = true;
  while (valid) {
    cout << "\t" << "Enter width: ";
    cin >> w;
    if (w >= 0) {
      width = w;
    }
    else {
      cout << "You have entered a negative number, inputs must be
positive."
      << endl;
      valid = false;
      break;
    }
    cout << "\t" <<  "Enter height: ";
    cin >> h;
    if (h >= 0) {
      height = h;
    }
    else {
      cout << "You have entered a negative number, inputs must be
positive."
      << endl;
      valid = false;
      break;
    }
    cout << "\t" <<  "Enter depth: ";
    cin >> d;
    if (d >= 0) {
      depth = d;
    }
    else {
      cout << "You have entered a negative number, inputs must be
positive."
      << endl;
      valid = false;
      break;
    }
    valid = false;
  }
}

void Furniture::print() {
  cout << name << ":" << endl;
  cout <<  "\t" << "Width = " << width << ", height = " << height << ",
depth = "
      << depth << endl;
}
```

```
/*
 *  @file   Bed.cpp
 *  @author Matthew Springer
 *  @date   February 7, 2016
 */

#include <iostream>
#include <string>

#include "Bed.h"

using namespace std;

// constructor
Bed::Bed(string nm, string sz) : Furniture(nm) {
      if ((sz.compare("Twin") == 0) ||
            (sz.compare("Full") == 0) ||
            (sz.compare("Queen") == 0) ||
            (sz.compare("King") == 0))
      {
            bedSize = sz;
            Bed::readDimensions();
      }
      else {
            cout << "\t" << "Bed size must be one of: 'Twin', 'Full',
'Queen', or 'King'"
            << endl;
      }
}

// destructor
Bed::~Bed() {}

/*
*  Function: print
*
*  Purpose:  print information about the bed to stdout
*/
void Bed::print() {
      Furniture::print();
      cout <<  "\t" << bedSize << " size" << endl;
}
```

```
/*
 *  @file   Table.cpp
 *  @author Matthew Springer
 *  @date   February 7, 2016
 */

#include <iostream>
#include <string>

#include "Table.h"

using namespace std;

// constructor
Table::Table(string nm, string wd) : Furniture(nm) {
      if ((wd.compare("Pine") == 0) ||
            (wd.compare("Oak") == 0) ||
            (wd.compare("Queen") == 0) ||
            (wd.compare("King") == 0))
      {
            wood = wd;
            Table::readDimensions();
      }
      else {
            cout << "\t" << "Wood type must be one of: 'Pine' or 'Oak'"
            << endl;
      }
}

// destructor
Table::~Table() {}

/*
*  Function: print
*
*  Purpose:  print information about the table to stdout
*/
void Table::print() {
      Furniture::print();
      cout <<  "\t" << wood << " wood" << endl;
}
```

```cpp
/*
 *  @file   Main.cpp
 *  @author Matthew Springer
 *  @date   February 7, 2016
 */


#include <iostream>
#include <string>

#include "Bed.h"
#include "Table.h"

using namespace std;

int main() {
	cout << "Creating table..." << endl;
	string tbl_name, wd_type;
	cout << "\t" << "Enter name: ";
	cin >> tbl_name;
	cout << "\t" << "Enter wood type (Pine, Oak): ";
	cin >> wd_type;
	Table new_table = Table(tbl_name, wd_type);

	cout << "Creating bed..." << endl;
	string bed_name, bed_size;
	cout << "\t" << "Enter name: ";
	cin >> bed_name;
	cout << "\t" << "Enter size (Twin, Full, Queen, King): ";
	cin >> bed_size;
	Bed new_bed = Bed(bed_name, bed_size);

	cout << endl << "Printing objects ..." << endl << endl;

	new_table.print();
	new_bed.print();
}
```

```
** CONSOLE OUTPUT **

Matts-MacBook-Pro:assign2 mspringer$ make
g++ -Wall -c -g Main.cpp
g++ -Wall -c -g Furniture.cpp
g++ -Wall -c -g Table.cpp
g++ -Wall -c -g Bed.cpp
g++ -Wall -g Main.o Furniture.o Table.o Bed.o -o Main
Matts-MacBook-Pro:assign2 mspringer$ ./Main
Creating table...
      Enter name: table1
      Enter wood type (Pine, Oak): Oak
      Enter width: 4
      Enter height: 5
      Enter depth: 7
Creating bed...
      Enter name: bed1
      Enter size (Twin, Full, Queen, King): Full
      Enter width: 5
      Enter height: 3
      Enter depth: 8

Printing objects ...

table1:
      Width = 4, height = 5, depth = 7
      Oak wood
bed1:
      Width = 5, height = 3, depth = 8
      Full size
```