Homework #2
Due: Friday, March 18th (11:59pm on Blackboard)

Programming: Follow proper coding standards as noted in our standards document. *Submit a zipped folder with: a) a single PDF with all your programs and output copied and pasted in to the file, and b) all your C++ programs including your input file for problem 1.*

1. (25 points) Download and use the included "CarRecords.txt" file from Blackboard as your input file. The file has 10 random used car records. Each record has 4 fields with the following format: car make, car model, year, color e.g.

```
Toyota, Matrix, 2006, silver
Volvo, XC70, 2009, blue
          .
         etc
```

Write an object-oriented C++ program in a single file called `CarRecords.cpp`. In the file, define two independent classes; `class Car`, `class CarRecords,` and the `main` function.

Implement `class Car` with the following description:

Car	

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

`public:`	`Car()`
	`setFields(string mk, string md, int yr,string cl)`
	`string getMake()`
	`string getModel()`
	`int getYear()`
	`string getColor()`

Variables:
`make, model, year, color` – the four data fields from the file

Methods:
`Car()` – default constructor sets the data fields to their default values.
`setFields(``string mk, string md, int yr,string cl`**`)`** – sets the class member fields to the values passed.
`string getMake()` – returns make
`string getModel()` – returns model
`string getYear()` – returns year
`string getColor()` – returns color

Implement `class CarRecords` with the following description:

CarRecords	

`private:`	`int arraySize   // keep track of number of records`
	`ifsteam infile`
	`Car *cars`

`public:`	`CarRecords(int size) // Reads file records to array`
	`~CarRecords()`
	`void printCarRecords ()`
	`void sort_cars_by_make()`
	`void sort_cars_by_year()`
	`void print_duplicates()`

Variables:

`arraySize` – size of the array set to the number of records read from the file
`infile` – input file stream used to read information from a file
`cars` – a pointer object (to a dynamically allocated array of objects)

Methods:

`CarRecords(int size)` – constructor sets the value passed to the `arraySize` member, creates a dynamic array enough to hold `arraySize` objects of type `Car`. In addition, the constructor uses the `infile` file stream and the `setFields()` method to initialize all the `cars` array elements with the car records read from the file.
`~CarRecords()` – Destructor frees the memory allocated with `new`, and closes the file handler.
`void printCarRecords()` – prints out the car records from the array of objects (see sample output)
`void sort_cars_by_make()` – sorts the records in ascending order based on the **make** field.
`void sort_cars_by_year()` – sorts the records in descending order based on the **year** field.
`void print_duplicates()` – identifies any repeated records, and prints them out when found. Repeated records means that all the fields are the same.

Test your program with the main program below. If the user enters a value equal to or less than 10 for records to read, the program should create an arrays with that many elements dynamically and should only read that many records from the file, e.g. if the user enters 5, the program should create an array of 5 elements and only read 5 car records. If the user enters a number greater than 10, the program should create an array of only 10 elements and read all 10 records from the file. This checking is done in the `CarRecords(int size)` constructor when it reads the file.

Note: For modularity, or if needed, your `classes` and the `main` function can have other helper functions. Your code should be well commented.

```
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->sort_cars_by_year();
      // Print car records
      cr->printCarRecords();
      // Sort by Make
      cr->sort_cars_by_make();
      // Print car records
      cr->printCarRecords();
      // Check for Duplicates
      cr->print_duplicates();
      delete cr;
} // end main
```

Sample output

```
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.....

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.....

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

2. (25 points) Develop and implement an object-oriented design that provides 3 different types of operations (arithmetic, logical, and relational comparison) on 3 different types of data (e.g., integers, floats, and doubles). To make your code reusable using templates, create a template class `Calculator` in two files (header file and source file) with the following description:

Calculator	

`private:`	`T value1`
	`T value2`

`Public:`	`Calculator()      // Initializes values to default`
	`Calculator(T value1, T value2) //set values`
	`T getValue1()      // Returns value1`
	`T getValue2()      // Returns value2`
	`T getSum()    // Uses arithmetic operator '+'`
	`int getLogicalAND() // Uses logical operator '&&'`
	`bool isGreater()   // Uses Relational operator '>'`

Create another file `CalcMain.cpp` that has another `class Arithmetic` in addition to the `main` function. The class `Arithmetic`, with the description below, will need class `Calculator` (using `#include` directive, NOT inheritance!).

Arithmetic	

`private:`	`int intData`
	`float floatData`
	`double doubleData`
	`void printOperations(T2 obj) //make this a`
	`                             //template function`

`Public:`	`Arithmetic()`
	`Arithmetic(int i, float f, double d)`
	`void intOperations(Arithmetic obj)`
	`void floatOperations(Arithmetic obj)`
	`void doubleOperations(Arithmetic obj)`

Variables:

`intData, floatData, doubleData` – an int, a float, and a double data members

Methods:

`printOperations(T2 obj)` – a template function local to this class. The function can be called with different types of objects, and it prints out the values and calls `getSum()`, `getLogicalAND()`, and `isGreater()` functions from the template class to test the 3 different types of operations, and print the results from those operations as seen in the sample output below.

`Arithmetic()` – a constructor initializes data members to zero.

`Arithmetic(int i, float f, double d)` – another constructor sets data members to values passed.

`intOperations(Arithmetic obj)` – creates an object of type `int` from the template class while setting the `int` values for the two objects, and calls `printOperations(T2 obj)` function to test the 3 different types of operations.

`floatOperations(Arithmetic obj)` – creates an object of type `float` from the template class while setting the `float` values for the two objects, and calls `printOperations(T2 obj)` function to test the 3 different types of operations.

`doubleOperations(Arithmetic obj)` – creates an object of type `double` from the template class while setting the `double` values for the two objects, and calls `printOperations(T2 obj)` function to test the 3 different types of operations.

Create a Makefile to compile your programs. Demonstrate that your program and your template class can operate of different data types. Provide a sample run for your output with different values for the `int`, `float`, and `double` values in `main` function.

To help you with the development, use this main function exactly as given below.
A Sample output might look like this:

```
Printing INTEGER operations...
4 + 7 = 11
4 && 7 = 1
4 > 7 = false

Printing FLOAT operations...
10.4 + 0 = 10.4
10.4 && 0 = 0
10.4 > 0 = true

Printing DOUBLE operations...
20.1 + 3.5 = 23.6
20.1 && 3.5 = 1
20.1 > 3.5 = true
```

Use this main function to test your progam

```
int main()
{
	// Create 1st object
	int int1 = 4;
	float f1 = 10.4;
	double d1 = 20.1;
	Arithmetic object1(int1, f1, d1);

	// Create 2nd object
	int int2 = 7;
	float f2 = 0.0;
	double d2 = 3.5;
	Arithmetic object2(int2, f2, d2);

	// Check operation on integer data type
	cout << "\nPrinting INTEGER operations..." << endl;
	object1.intOperations(object2);

	// Check operation on float data type
	cout << "\nPrinting FLOAT operations..." << endl;
	object1.floatOperations(object2);

	// Check operation on double data type
	cout << "\nPrinting DOUBLE operations..." << endl;
	object1.doubleOperations(object2);
} // end main
```

Note: For modularity, or if needed, your `classes` and the `main` function can have other helper functions. Your code should be well commented.

3. (20 points) Define a class hierarchy formed of a parent class `Furniture` and two child classes `Table` and `Bed`. The parent class should contain the following members:
   - Three private floating-point numbers indicating the dimensions of the piece of furniture (`width`, `height`, and `depth`).
   - A private string containing a unique name.
   - A public constructor that takes the name of the piece of furniture as an argument.
   - A public function `ReadDimentions`, used to read the values of the width, height, and depth from the keyboard. The function should show an error message if any of the entered values is less than 0.
   - A public virtual function `Print()`, with no arguments and a `void` return value. This function should print the dimensions and name.
   - The `Table` child class should have the following members:
   - A private string indicating the type of wood used for the table, with two possible valid values: `“Pine”` and `“Oak”`.
   - A public constructor that takes the unique name of the table as the first argument, passed

directly to the constructor of the parent class, and the string corresponding to the wood type as the second argument. The constructor should print an error message if an invalid size string is passed.

- A public function `Print()` that overrides the function with the same name in the parent class. This function prints common furniture information by invoking the parent class function, and continues printing information specific to a table (the wood type).

The `Bed` child class should have the following members:

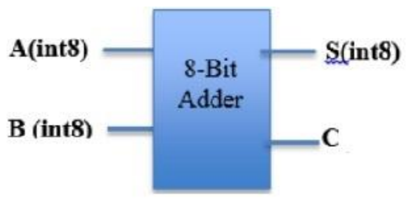
- A private string field specifying the size of the bed, with four possible valid values: `“Twin”`, `“Full”`, `“Queen”`, and `“King”`.
- A public constructor that takes the unique name of the bed as the first argument, passed directly to the constructor of the parent class, and the string corresponding to the bed size as the second argument. The constructor should print an error message if an invalid size string is passed.
- A public function `Print()` that overrides the function with the same name in the parent class. This function prints common furniture information by invoking the parent class function, and continues printing information specific to a bed (the size string).

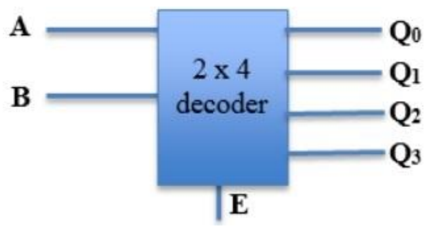
Declare and define **each** of the three classes with a header (.h) file and a corresponding source (.cpp) file, and an additional file `main.cpp`, which will include the main program. The main program should instantiate one object of type `Table` and another object of type `Bed`. Its properties should be populated based on values read from the user, and then printed with invocations to functions `Print()`.

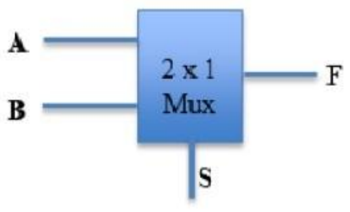
This is an example of the execution of the program

```
Creating table...
  Enter name: MyTable
  Enter wood type (Pine, Oak): Pine
  Enter width: 1
  Enter height: 2
  Enter depth: 3
Creating bed...
  Enter name: MyBed
  Enter size (Twin, Full, Queen, King): Queen
  Enter width: 5
  Enter height: 6
  Enter depth: 7

Printing objects ...

MyTable:
  Width = 1, height = 2, depth = 3
  Pine wood
MyBed:
  Width = 5, height = 6, depth = 7
  Queen size
```

4. (15 points) An 8-bit adder (designed in lab5), a 2-to-4 decoder, and a 2-to-1 multiplexor are shown as follows. Using these blocks, and any other necessary logic gates:







a.) Design a 16-bit adder out of 8-bit adders. Your 16-bit adder should have 2 `int16` data types to be added, and 1 `int16` data type sum and a carry out bit.

b.) Design a 3-to-8 decoder out of 2-to-4 decoders

c.) Design a 4-to-1 multiplexor out of 2-to-1 multiplexors

5. (15 points) Design a simple arithmetic and logic unit (ALU) that takes as input two operation selection lines (s0 and s1), and two 2-bit data lines A[1:0] and B[1:0]. The output of the circuit is C[1:0]. The values of the input selection lines are: 00 – shift the first operand right by 1 position, 01 - shift the second operand left 1 position, 10 – add the two 2-bit quantities, 11 – set the output to zero.

   Input your design in Simulink and print out the resulting combinational circuit diagram. Also provide screen shots of running simulation of this design in Simulink.