## CIS 22B Assignment 5

Use the same format for problem and function headings as assignment 1.

Copy the solution from problem 4.2

## Problem 5.1

In this problem we will use inheritance to create two new classes, both of which will inherit from the class Car

Do not change the StringOfCar class .   
You are not using a StringOfCars in this problem, but will need it later.   
You can remove the StringOfCars from the parameter list for the input function and put it back later,  
or keep the StringOfCars parameter, pass a StringOfCars to the input funcion, and ignore it.

As a preliminary step, create a new global function (that is a function that is not a member function) named buildCar.   
The buildCar function has the five parameters needed for a Car.   
When called, it builds an object of type Car by using the Car constructor that has five parameters.   
For testing in this assignment, after building the car object, the buildCar function calls the output member function for the Car.   
Now, modify the input function so it calls the buildCar function to build the car, rather than building the car in the input function.

The kind of cars for the three classes will be:   
Car:   business, maintenance, tender, other   
FreightCar:   box, tank, flat, hopper, otherFreight   
PassengerCar:   chair, sleeper, otherPassenger, combine

Use an **enum** to keep the kind, rather than using a string as we did in previous problems.   
In the global area define an enum named Kind, with the following values in this order:   
business, maintenance, tender, other box, tank, flat, hopper, otherFreight, chair, sleeper, otherPassenger, combine.   
Also in the global area define an array of const string objects named KIND\_ARRAY.   
It will contain strings with the same text as the names of the values in the enum, in the same order.

Change the Kind field in the Car class so it is: Kind kind;   
Adjust the functions to use the enum and KIND\_ARRAY.   
The output function can use the KIND\_ARRAY values to print the string corresponding to the enum value.

Build a new member function setKind. in the Car::setUpCar function, pass it a constant reference to the string provided by the user.   
If the string is not business, maintenance or tender in the Car class, set the kind to other.   
The setKind member function will set the correct Kind.

Change private to protected in the Car class only.

Make two classes that inherit from the Car class:   FreightCar and PassengerCar.   
Each class will need a default constructor, a copy constructor, and a constructor that has five parameters.   
Only one more function will be built in each class; all the rest will be inherited.   
No additional member data will be added.

Create setKind functions for the FreightCar and PassengerCar classes that are similar to the setKind function for the Car class, but with different values.   
The setKind function for the FreightCar class uses only the values:   box, tank, flat ,hopper, otherFreight   
The setKind function for the PassengerCar class uses only the values:   chair, sleeper, otherPassenger, combine

Create two new global function named buildFreightCar and buildPassengerCar, similar to the buildCar function.   
These are used to build a FreightCar or a Passenger car, respectively.

Revise the main function to call the input function and do nothing else.

In the input function loop read one line from the file each time through the loop, look at the Type field in the record and call the corresponding build function to build that type of car.   
Before calling the appropriate build function, print a header giving the sequence number of the car read, with the number 1 for the first car and incremented for each successive car.

The file for this problem should contain (without headers, of course):

Type order ARR number kind loaded destination

Car car1 SP 819480 maintenance false NONE

Car car2 SLSF 46870 business true Memphis

Car car3 AOK 150 tender true McAlester

FreightCar car4 MKT 123450 tank false Fort Worth

FreightCar car5 MP 98760 box true Saint Louis

FreightCar car6 SP 567890 flat true Chicago

FreightCar car7 GMO 7870 hopper true Mobile

PassengerCar car8 KCS 7890 chair true Kansas City

PassengerCar car9 PAPX 140 sleeper true Tucson

PassengerCar car10 B&O 740 combine false NONE

This file is provided as cardata51.txt

## Problem 5.2

Copy the solution from problem 5.1.

In this problem you will change the StringOfCars class so it has an array of pointers to objects, rather than an array of objects themselves. This will allow us to have a string of cars that contains Car, FreightCar, and PassengerCar objects all in the same string of cars. This works because a pointer of type Car \* can be made to point to Car objects as well as point to the child FreightCar and PassengerCar objects.

Remove the call to the output member function from the three build funtions:   buildCar, buildFreightCar, and buildPassengerCar.

Because you have pointers of type Car \* that may point to any one of the three types of objects, there is a problem. The system does not know what type object will be encountered until execution time. That means a system is needed so the functions that are overridden need to have a mechanism to select the correct version of the function at execution time, rather than having it fixed at compile time. This is done with the virtual declaration. To do this make the declaration of the setKind and the declaration of the ~Car functions **virtual** in the Car class.   
This is only done in the declaration, not the definition of the function.   
This is only done in the parent class, not the children classes.

To change the class StringOfCars, comment out all the code in the member functions of the StringOfCars class and fix them one or two at a time in the following order. These are similar to the previous functions, but changed to allow for the fact that we are putting pointers to cars in the array.

* Build the default constructor first. Create a default string of cars in main.
* Build an output function, similar to the old one, but dereferrencing the pointers.
* Write a push function which adds a car to the string of cars. It takes a Car by constant reference, allocates space in the heap, makes a copy of the Car, and puts the pointer to the Car in the array.
* Write a copy constructor similar to the old one, but it gets space for each car and copies each one, as well as getting space for the array.
* omit the pop member function.

Add to the build functions a call to push the objects onto the string of cars.

Remove the output from the build functions.

Test the copy constructor by making stringOfCars2 in the stack for main that is a copy of stringOfCars1.

Print stringOfCars2.