

Name: Garrett Jackson

More about Classes

Problems to do in Lab. Do not report this.

Problem 1. Bank Program

Define the class **Bank** to hold the basic information to describe a bank with member variables: Bank name (string), Bank type (enum, commercial/thrifts/credit union), Bank address (string). Add appropriate member functions to manipulate an object.

Define the class **Account** to implement the basic properties of a bank account with member variables: Account ID (int), Account holder's name (string), account type (enum, checking/saving), and balance (double). Add appropriate member functions to manipulate an object. Use a static member in the class to automatically assign Account IDs.

Also, define a friend function for both the Bank class and the Account class. The friend function should display the Bank details and an array of 5 components of type Account to process up to 5 customers. Write a separate program (different file) to illustrate how to use your classes and friend function.

Problem 2. Implementing a String class

Create a custom String class that mimics basic string behavior using dynamic memory allocation. Your class should store characters using a dynamically allocated char* array.

Implement:

- A constructor to initialize a string from a C-style string (const char*).
- A copy constructor to handle deep copy.
- A destructor to release allocated memory.
- An overloaded assignment operator (operator=) to assign one String object to another safely.
- An overloaded + operator to concatenate two String objects and return a new one.

To be reported on canvas. Create a PDF. Include screenshots of code and execution. Include copy-pasteable text of code. Be careful with variable names and indentation. You must use the templates.

Single problem: 20 points.

Problem 3. Car Instrument Simulator

Design a set of classes that work together to simulate a car's fuel gauge and odometer.

The FuelGauge class will store the car's current amount of fuel in gallons, report the car's current amount of fuel, indicate when the fuel tank is empty, and refuel the car's fuel tank to a maximum of 15 gallons. The Odometer class will be a friend of the FuelGauge class, so it will have access to private members of the FuelGauge class.

The Odometer class will store the car's current mileage, and report the car's current mileage. As well, it will increment the mileage by 1 mile. When the maximum mileage of 999,999 miles is exceeded it will reset the current mileage to 0. For every 24 miles, the Odometer class will decrease the FuelGauge object by 1 gallon to simulate the car's fuel economy as 24 miles per gallon.

Demonstrate the classes by creating instances of each. Simulate filling the car up with fuel, and then run a loop that increments the odometer until the car runs out of fuel. During each loop iteration, print the car's current mileage and amount of fuel.

USE THE NEXT TEMPLATE (MANDATORY) FOR THE MAIN FUNCTION

```
//DO NOT MODIFY THIS SECTION
#include <iostream>
#include "FuelGauge.h"
#include "Odometer.h"
using namespace std;

int main()
{
    //ifstream ifile;
    Odometer odom( 999990 );    //Create and initialize
    FuelGauge gauge( 0 );      //Create and initialize to zero
    odom.report();              //Display info from odom
    gauge.report();             //Display info from gauge
    gauge.addToTank( 1 );      //Add one gallon
    gauge.report();             //Display new info from gauge
    while( gauge.getGallons()>0 ) //Repeat while we still have
fuel
    {
        odom.advance( 1, gauge ); //Advance one mile. Include
reference to gauge
        odom.report();             //Report info from odom
        gauge.report();            //Report info from gauge
    }
    return 0;
}
```

Compare **main()** with the output to understand the operation.

Output:

```
Mileage: 999990
Fuel: 0
```

EMPTY!

Adding fuel, going from 0 to 1

Fuel: 1

Mileage: 999991

Fuel: 1

Mileage: 999992

Fuel: 1

Mileage: 999993

Fuel: 1

Mileage: 999994

Fuel: 1

Mileage: 999995

Fuel: 1

Mileage: 999996

Fuel: 1

Mileage: 999997

Fuel: 1

Mileage: 999998

Fuel: 1

Mileage: 999999

Fuel: 1

Mileage: 0

Fuel: 1

Mileage: 1

Fuel: 1

Mileage: 2

Fuel: 1

Mileage: 3

Fuel: 1

Mileage: 4

Fuel: 1

Mileage: 5

Fuel: 1

Mileage: 6

Fuel: 1

Mileage: 7

Fuel: 1

Mileage: 8

Fuel: 1

Mileage: 9

Fuel: 1

Mileage: 10

Fuel: 1

Mileage: 11

Fuel: 1

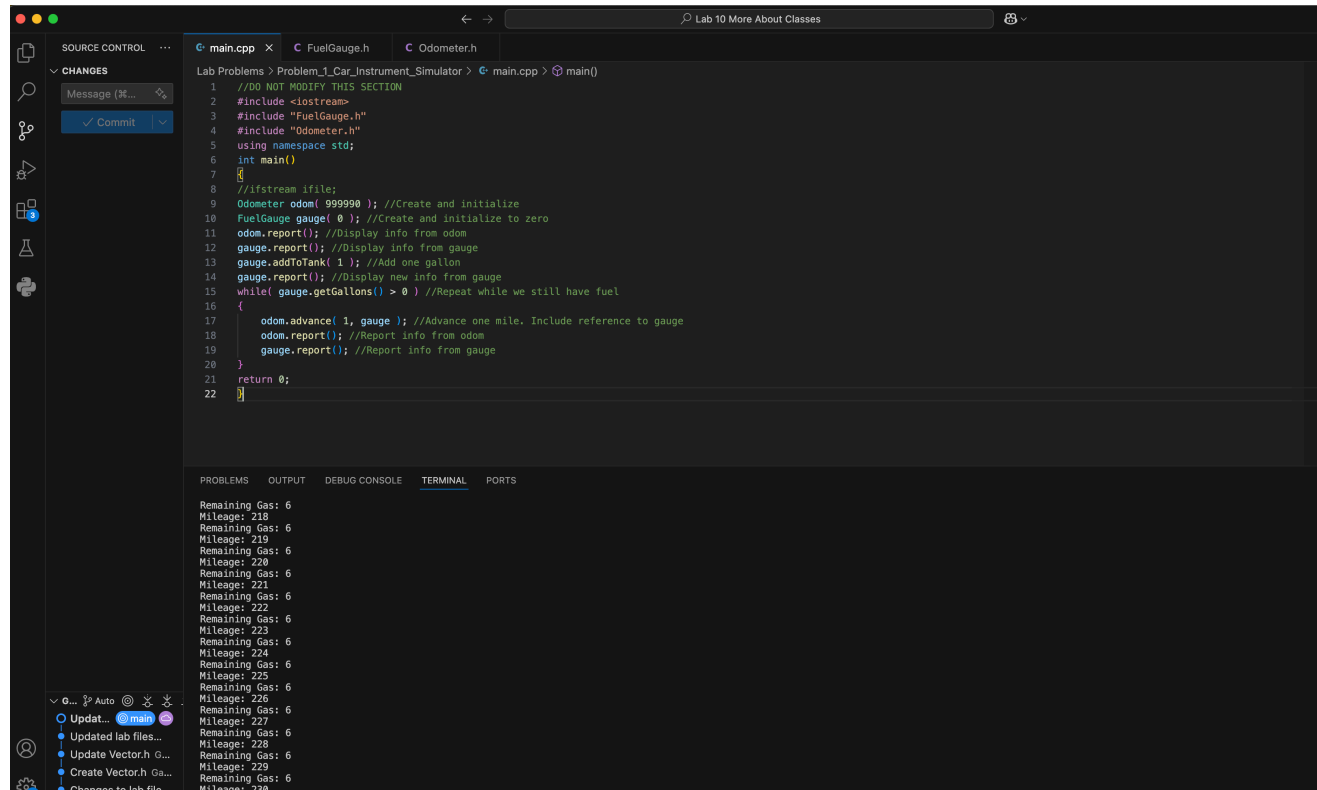
Mileage: 12

Fuel: 1

Mileage: 13

Fuel: 1

Mileage: 14
Fuel: 0
EMPTY!



```
1 //DO NOT MODIFY THIS SECTION
2 #include <iostream>
3 #include "FuelGauge.h"
4 #include "Odometer.h"
5 using namespace std;
6 int main()
7 {
8     //ifstream ifile;
9     Odometer odom( 999990 ); //Create and initialize
10    FuelGauge gauge( 0 ); //Create and initialize to zero
11    odom.report(); //Display info from odom
12    gauge.report(); //Display info from gauge
13    gauge.addToTank( 1 ); //Add one gallon
14    gauge.report(); //Display new info from gauge
15    while( gauge.getGallons() > 0 ) //Repeat while we still have fuel
16    {
17        odom.advance( 1, gauge ); //Advance one mile. Include reference to gauge
18        odom.report(); //Report info from odom
19        gauge.report(); //Report info from gauge
20    }
21    return 0;
22 }
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

Remaining Gas: 6
Mileage: 218
Remaining Gas: 6
Mileage: 219
Remaining Gas: 6
Mileage: 220
Remaining Gas: 6
Mileage: 221
Remaining Gas: 6
Mileage: 222
Remaining Gas: 6
Mileage: 223
Remaining Gas: 6
Mileage: 224
Remaining Gas: 6
Mileage: 225
Remaining Gas: 6
Mileage: 226
Remaining Gas: 6
Mileage: 227
Remaining Gas: 6
Mileage: 228
Remaining Gas: 6
Mileage: 229
Remaining Gas: 6
Mileage: 230

```
//DO NOT MODIFY THIS SECTION
```

```
#include <iostream>
```

```
#include "FuelGauge.h"
```

```
#include "Odometer.h"
```

```
using namespace std;
```

```
int main()
```

```
{
```

```
//ifstream ifile;
```

```
Odometer odom( 999990 ); //Create and initialize
```

```
FuelGauge gauge( 0 ); //Create and initialize to zero
```

```

odom.report(); //Display info from odom

gauge.report(); //Display info from gauge

gauge.addToTank( 1 ); //Add one gallon

gauge.report(); //Display new info from gauge

while( gauge.getGallons() > 0 ) //Repeat while we still have fuel
{

    odom.advance( 1, gauge ); //Advance one mile. Include reference
    to gauge

    odom.report(); //Report info from odom

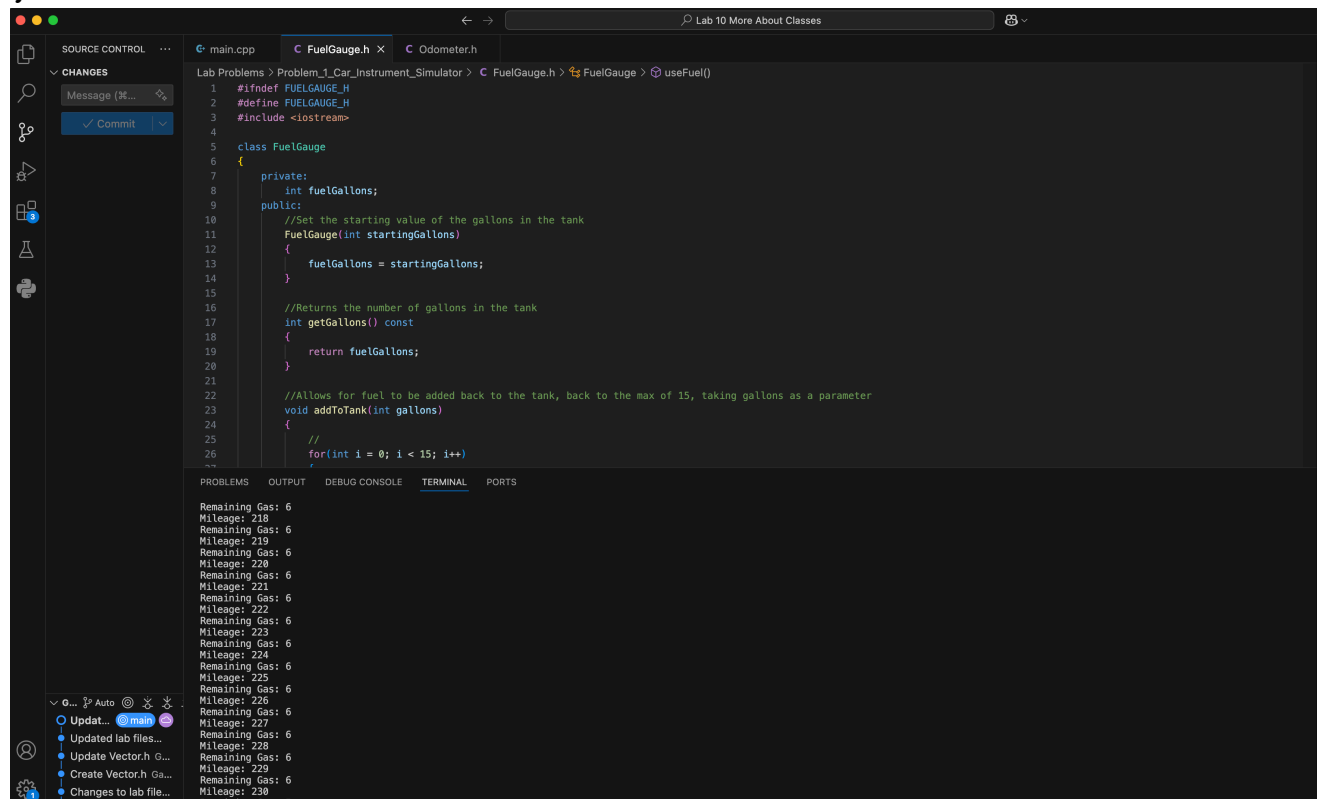
    gauge.report(); //Report info from gauge

}

return 0;

}

```



```
#ifndef FUELGAUGE_H
```

```
#define FUELGAUGE_H

#include <iostream>

class FuelGauge
{
    private:
        int fuelGallons;

    public:
        //Set the starting value of the gallons in the tank
        FuelGauge(int startingGallons)
        {
            fuelGallons = startingGallons;
        }

        //Returns the number of gallons in the tank
        int getGallons() const
        {
            return fuelGallons;
        }

        //Allows for fuel to be added back to the tank, back to the
        max of 15, taking gallons as a parameter
        void addToTank(int gallons)
        {
            //
```

```
        for(int i = 0; i < 15; i++)
        {
            // Checks if the private member plus the parameter
is less than 15
            if(fuelGallons + gallons <= 15)
            {
                // Adds the parameter to the private member
                fuelGallons += gallons;
            }
            else if(fuelGallons == 0)
            {
                fuelGallons = 15;
            }
        }
    }
```

//Function for fuel to be subtracted from the car's tank

```
void useFuel()
```

```
{
    //Decrement by one as long as the fuel is above zero
    if(fuelGallons > 0)
    {
        fuelGallons--;
    }

    // Once the tank reaches zero, state the tank is empty
```

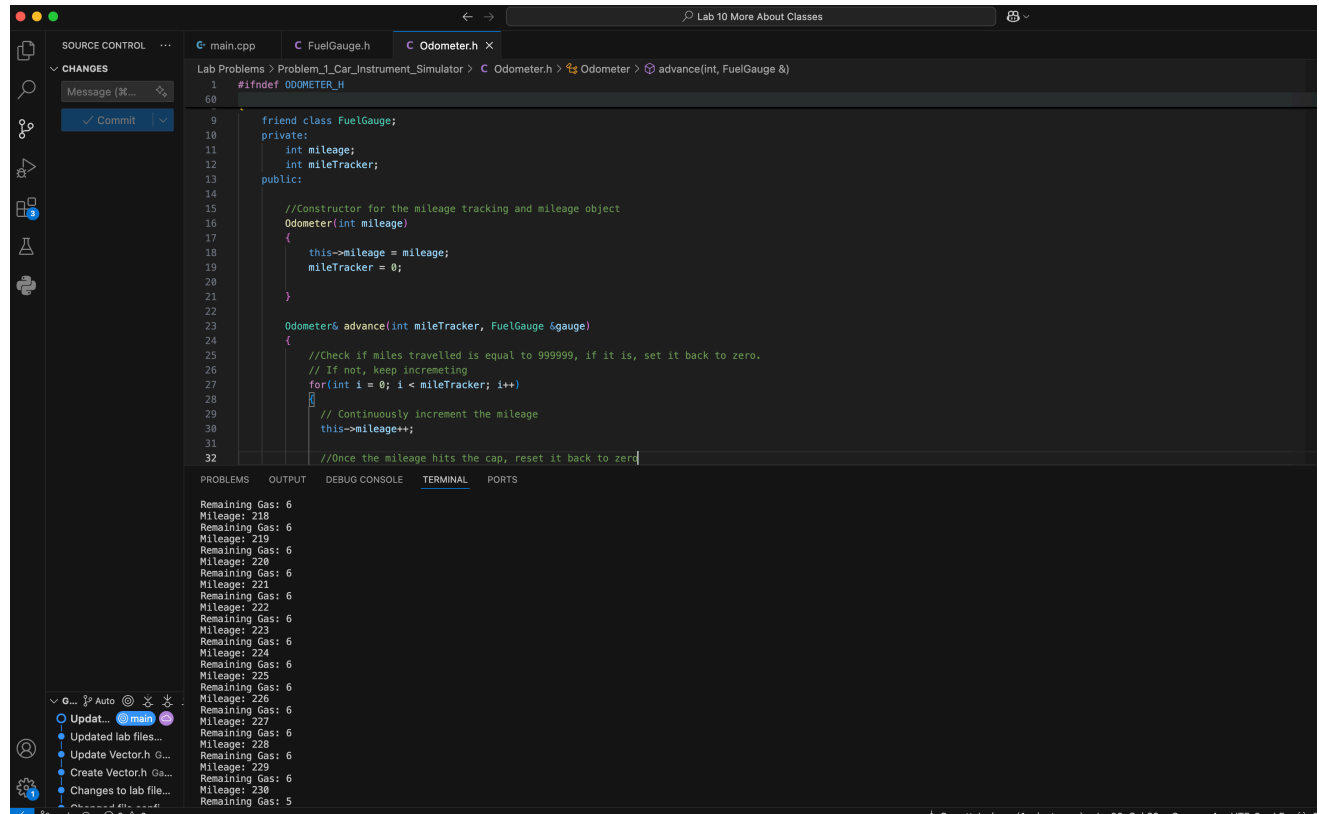
```
        if(fuelGallons == 0)
        {
            std::cout << "Tank empty. Please refill" <<
std::endl;
        }

    }

    //Display's the remaining gas when called
    void report()
    {
        std::cout << "Remaining Gas: " << fuelGallons <<
std::endl;
    }

};
```


#endif



#ifndef ODOMETER_H

#define ODOMETER_H

#include <iostream>

class FuelGauge;

class Odometer

{

friend class FuelGauge;

private:

int mileage;

int mileTracker;

```
public:

    //Constructor for the mileage tracking and mileage object
    Odometer(int mileage)
    {
        this->mileage = mileage;
        mileTracker = 0;
    }

    Odometer& advance(int mileTracker, FuelGauge &gauge)
    {
        //Check if miles travelled is equal to 999999, if it
        //is, set it back to zero.

        // If not, keep incrementing
        for(int i = 0; i < mileTracker; i++)
        {
            // Continuously increment the mileage
            this->mileage++;

            //Once the mileage hits the cap, reset it back to
            //zero

            if(this->mileage > 999999)
            {
                this->mileage = 0;
            }
        }
    }
};
```

```
    }

    //Continuously increment the tracker

    this->mileTracker++;

    //Checks when the mileage equals 24, then utilizes
the useFuel function from the FuelGauge Class

    // Then sets the tracker back to 0
    if(this->mileTracker == 24)
    {
        gauge.useFuel();

        this->mileTracker = 0;
    }
}

return *this;

}

void report()
{
    std::cout << "Mileage: " << mileage << std::endl;
}

};
```

#endif