Final Exam

Parts of it have been submitted via myclasses they are noted below

Jeremy Scheuerman

Part 1:Submitted via MyClasses

Part 2

Q1-Uml Diagram :Submitted Via my classes

Q2-Implementation (only the first couble onjects are actually complete but it shows derived class inheritance and use of virtual function so I get the idea, just ran out of time)

```
#include <iostream>


using namespace std;
class Transportation
{
private:
   string owner;
   int year;
public:
   Transportation(string o,int y);
   void displayOwner();
   void displayYear();
   virtual void displayInfo();
};
Transportation::Transportation(string o,int y)
{
   owner=o;
```

```
    year=y;
}
void Transportation::displayYear()
{
    cout<<"Year: "<<year<<endl;
}
void Transportation::displayOwner()
{
    cout<<"Owner: "<<owner<<endl;
}
void Transportation::displayInfo()
{
    displayOwner();
    displayYear();
}
class Airplane:public Transportation
{
private:
    double altitude;
public:
    Airplane(string o,int y,double a);
    void displayAltitude();
    virtual void displayInfo();
};
Airplane::Airplane(string o,int y,double a):Transportation(o,y)
//derive and use overload constructor
```

```
{
   altitude=a;
}
void Airplane::displayAltitude()
{
   cout<<"Altitude: "<<altitude<<endl;
}
void Airplane::displayInfo()
{
   displayOwner();
   displayYear();
   displayAltitude();
}
class Helicopter:public Airplane
{
private:
   int propellers;
public :
   Helicopter(string o,int y,double a, int p);
   void displayPropellers();
   virtual void displayInfo();


};
class Jetplane:public Airplane
{
private:
    string airline;
public :
    Jetplane(string o,int y,double a, string airl);
    void displayAirline();
    virtual void displayInfo();


};
class Boat:public Transportation
{
private:
    int motors;
public:
    void displayMotors();
    virtual void displayInfo();
};
class SpeedBoat:public Boat
{
private :
    double topSpeed;
public:
    void displayTopSpeed();


};
class CruiseBoat:public Boat
{
private :
    int capacity;
public:
    void displayCapacity();


};
class Vehicle(string type,string model):public Transportation
{
private:
    string type;
    string model;
public:
    virtual void displayInfo();
};
class Car:public Vehicle
{
private:
    int fuelEff;
public:
    void displayfuelEff ());
    virtual void displayInfo();
};
class Truck:public Vehicle
{
private:
    string color;
    string wheels;
```

```
public:
    void displayWheels();
    void displayColor();
    virtual void displayInfo();
};
```

Q3-Main

```
int main()
{
    //main to show virtual functions with dynamic binding\
    //trasnp
    Transportation trans("Jeremy",2000);
    cout<<"Transporation"<<endl;
    trans.displayInfo();
    //planes
    Airplane airp("Billy",2000,5600);
    cout<<"Airplane"<<endl;
    airp.displayInfo();
    Helicopter heli("Jack",2005,4600);
    cout<<"Helicopter"<<endl;
    heli.displayInfo();
    Jetplane jetp("Howard",2011,8700);
    cout<<"Jetplane"<<endl;
    jetp.displayInfo();
    //boats
    Boat boat("Charles",2003,2);
    cout<<"Boat"<<endl;
    boat.displayInfo();

    SpeedBoat speedboat("Mike",2006,3,80.7);

    cout<<"SpeedBoat"<<endl;

    speedboat.displayInfo();

    CruiseBoat cruiseboat("Tony",1998,6,500);

    cout<<"SpeedBoat"<<endl;

    cruiseboat.displayInfo();

    //vehicles

    Vehicle vehicle("Hailey",1995,"Honda","Accord");

    cout<<"Vehicle"<<endl;

    vehicle.displayInfo();

    Car car("Andy",2002,"Toyota","Rav4",17);

    cout<<"Car"<<endl;

    car.displayInfo();

    Truck truck("Kevin",2002,"Ford","f250","Green",6);

    cout<<"Vehicle"<<endl;

    truck.displayInfo();

    return 0;

}
```

Part 3

Q1:Submitted Via MyClasses

Q2:recursive function to count nodes

```
int countNodes(Node<T> head)

{

    Node<T>* curr=head;

//set curr to head
```

```
    int i=1;
    if (head==nullptr)
    {
        return 0;
    }
    else
    {
        i+=countNodes(curr->nextNode)
           //do recursion
           return i;
    }
}
```

Q3:RemoveDup

```
void removeDup(const list<T>& l1, const list<T>& l2, list<T>& l3)
{
    typename list<T>::iterator iter_1 = l1.begin();
    typename list<T>::iterator iter_2 = l2.begin();
    typename list<T>::iterator iter_3 = l3.begin();
    bool dupl = false;
    bool dupl_3 = false;
    //values to keep track of where dupes are
    for (iter_1=l1.begin(); iter_1!=l1.end()iter_1++)
    {
        for (iter_2=l2.begin(); iter_2!=l2.end()iter_2++)
        {
            if(*iter_1==*iter_2)
            {
//if there is a match
                dupl=true;
            }
        }
        if (dupl==false)
        {
            for (iter_3=l3.begin(); iter_3!=l3.end()iter_3++)
            {
                if (*iter_1==*iter_3)
                {
                    //there is a duplicate in 3
                    dupl_3=true;
                    break;
                }
            }
        }
        if (dupl_3==false)
        {
            //if no duplicate in 3
            l3.push_back(*iter_1);
            //add value from 1 to l3
        }
        dupl_3=false;
        dupl=false;
        //reset values
```

```
    }

    //do same process for the second list

    for (iter_2=l2.begin(); iter_2!=l2.end()iter_2++)

    {

        for (iter_1=l1.begin(); iter_1!=l1.end()iter_1++)

        {

            if(*iter_1==*iter_2)

            {

//if there is a match

                dupl=true;

            }

        }

        if (dupl==false)

        {

            for (iter_3=l3.begin(); iter_3!=l3.end()iter_3++)

            {

                if (*iter_2==*iter_3)

                {

                    //there is a duplicate in 3

                    dupl_3=true;

                    break;

                }

            }

        }

        if (dupl_3==false)

        {
            //if no duplicate in 3
            l3.push_back(*iter_2);
            //add value from 2 to l3
        }
        dupl_3=false;
        dupl=false;
        //reset values
    }
}
```

Q4:Submitted Via Myclasses

Q5:

Method 1

```
double operator +(circle circ_1,circle circ_2)
//overload
{
            double value=circ_1.area()+circ_2.area();
            return value;
}
double circle::area()
{
            double value=(radius*radius)*3.14;
            return value
}
```

Method 2

```
Double circle::operator+circle+(circle circ){
```

```
Double value=(radius*radius*3.14)+(circ.radius*circ.radius*3.14);

return value;

}

double operator+(circle circ_1,circ_2)

//overload

{

Double value=(circ_1.radius*circ_1.radius*3.14)+(circ_2.radius*circ_2.radius*3.14);

}
```

Method 3

```
Class Circle

{

Private:

//members

        float radius;

public:

//overload

        double operator+(circle c);

        friend float operator+(circle c);

}
```