CIS 200 - Project 03

1. Problem Statement

Create a program where a user takes a trip in a metaphorical vehicle (life?). These vehicles are derived from an original provided vehicle structure and will allow the user to perform varying actions on/in the vehicle during the trip.

2. Requirements

2.1 Assumptions

- The user will keep track of path traveled on paper
- Vehicle and some child classes are provided
- Command Line Input Only
- Vehicle movement will be handled like *Zork*

2.2 Specifications

- The user will choose a vehicle
- The user will choose actions to perform in the vehicle
- Actions will be logged to file
- Actions will be confirmed to the user
- Certain actions will require previous actions
- The user may switch vehicles starting a new trip
- The user will determine when the trip ends

3. Decomposition Diagram

- Program
    - Input
        - User Chooses Vehicle
        - User Chooses Action
        - User Ends Trip
    - Process
        - Determine if Action is Available/Applicable
        - *Perform* Action Within Vehicle
    - Output
        - Confirmation of Action to User
        - File Containing Actions Taken

4. Test Strategy

- File Existence
- Valid Vehicle
- Invalid Vehicle
- Valid Boat
- Invalid Boat
- Valid Plane
- Invalid Plane
- Valid Land Vehicle
- Invalid Land Vehicle
- Valid Car
- Invalid Car
- Valid Truck
- Invalid Truck
- Path

5. Test Plan Version 1

Test Strategy	#	Description	Input	Expected Output	Actual Output	Pass/Fail
File Existence	1	Does File Exist?				
Valid Vehicle	1	*setAge*()	Anything >= 0			
Valid Vehicle	2	*setPrice*()	Anything >= 0.00			
Valid Vehicle	3	*setHeading*()	“North”			
Valid Vehicle	4	*setHeading*()	“South”			
Valid Vehicle	5	*setHeading*()	“East”			
Valid Vehicle	6	*setHeading*()	“West”			
Valid Vehicle	7	*setIsStarted*()	TRUE			
Valid Vehicle	8	*setIsStarted*()	FALSE			
Valid Vehicle	9	*getAge*()	NULL			
Valid Vehicle	10	*getPrice*()	NULL			

Valid Vehicle	11	*moveNorth*()	NULL			
Valid Vehicle	12	*moveSouth*()	NULL			
Valid Vehicle	13	*moveEast*()	NULL			
Valid Vehicle	14	*moveWest*()	NULL			
Valid Vehicle	15	*printVehicleInfo*()	NULL			
Invalid Vehicle	1	*setAge*()	Anything < 0			
Invalid Vehicle	2	*setPrice*()	Anything < 0.00			
Valid Boat	1	*setIsDocked*()	TRUE			
Valid Boat	2	*setIsDocked*()	FALSE			
Valid Boat	3	*setFlagRaised*()	TRUE			
Valid Boat	4	*setFlagRaised*()	FALSE			
Valid Boat	5	*getIsDocked*()	NULL			
Valid Boat	6	*getIsDocked*()	NULL			
Valid Boat	7	*getFlagRaised*()	NULL			
Valid Boat	8	*getFlagRaised*()	NULL			
Valid Boat	9	*toggleFlag*()	FALSE			
Valid Boat	10	*toggleFlag*()	TRUE			
Valid Boat	11	*undock*()	Docked			
Valid Boat	12	*undock*()	Undocked			
Valid Boat	13	*dock*()	Undocked			
Valid Boat	14	*dock*()	Docked			
Valid Boat	13	*printBoatInfo*()	NULL			
Valid Plane	1	*setLiftoffStatus*()	TRUE			
Valid Plane	2	*setLiftoffStatus*()	FALSE			

Valid Plane	3	*setAltitude*()	Anything >= 0			
Valid Plane	4	*getLiftoffStatus*()	NULL			
Valid Plane	5	*getLiftoffStatus*()	NULL			
Valid Plane	6	*getAltitude*()	NULL			
Valid Plane	7	*getAltitude*()	NULL			
Valid Plane	8	*takeoff*()	Landed			
Valid Plane	9	*takeoff*()	Taken Off			
Valid Plane	10	*ascend*()	Taken Off			
Valid Plane	11	*ascend*()	Altitude 15			
Valid Plane	12	*ascend*()	Landed			
Valid Plane	13	*descend*()	Taken Off			
Valid Plane	14	*descend*()	Altitude 1			
Valid Plane	15	*descend*()	Landed			
Valid Plane	16	*land*()	Altitude > 0			
Valid Plane	17	*land*()	Altitude = 0			
Valid Plane	19	*printPlaneInfo*()	NULL			
Invalid Plane	1	*setAltitude*()	Anything < 0			
Valid Land V	1	*setNumPass…*()	0 < Anything < 5			
Valid Land V	2	*setIsParked*()	TRUE			
Valid Land V	3	*setIsParked*()	FALSE			
Valid Land V	4	*setWindshield..*()	TRUE			
Valdi Land V	5	*setWindshield..*()	FALSE			
Valid Land V	6	*getNumPass…*()	NULL			
Valid Land V	7	*getWindshield..*()	NULL			
Valid Land V	8	*getWindshield..*()	NULL			

Valid Land V	9	*toggleWinshiel..*()	Active			
Valid Land V	10	*toggleWinshiel..*()	Inactive			
Valid Land V	12	*park*()	*isParked* = T			
Valid Land V	13	*drive*()	*isParked* = F			
Valid Land V	14	*printLandVehi…*()	NULL			
Invalid Land V	1	*setNumPass…*()	X < 1 || X > 4			
Valid Car	1	*setRaceCar…*()	TRUE			
Valid Car	2	*setRaceCar…*()	FALSE			
Valid Car	3	*setRadioSt…*()	TRUE			
Valid Car	4	*setRadioSt…*()	FALSE			
Valid Car	5	*getRaceCar…*()	NULL			
Valid Car	6	*getRaceCar…*()	NULL			
Valid Car	7	*getRadioSt…*()	NULL			
Valid Car	8	*getRadioSt…*()	NULL			
Valid Car	9	*toggleRadio*()	On			
Valid Car	10	*toggleRadio*()	Off			
Valid Car	11	*toggleSpoiler*()	Up			
Valid Car	12	*toggleSpoiler*()	Down			
Valid Car	13	*printCarInfo*()	NULL			
Valid Truck	1	*setDieselType…*()	TRUE			
Valid Truck	2	*setDieselType…*()	FALSE			
Valid Truck	3	*setFwdStatus*()	TRUE			
Valid Truck	4	*setFwdStatus*()	FALSE			

Valid Truck	5	*getDieselTyp*…()	NULL			
Valid Truck	6	*getDieselTyp*…()	NULL			
Valid Truck	7	*getFwdStatus*()	NULL			
Valid Truck	8	*getFwdStatus*()	NULL			
Valid Truck	9	*toggleFWD*()	Off			
Valid Truck	10	*toggleFWD*()	ON			
Valid Truck	11	*exchangeCorn*…()	Not Diesel			
Valid Truck	12	*exchangeCorn*…()	Diesel			
Valid Truck	13	*printTruckInfo*()	NULL			

6. Initial Algorithm

1. Create Parent Class
    a. Name: *vehicle*
    b. Data:
        i. Integer *age*
        ii. Float *price*
        iii. String *heading*
        iv. Bool *isStarted*
    c. Functions:
        i. Void *setAge*(integer input)
            1. IF input >= 0
                a. *age* = input
            2. ELSE
                a. Print “INVALID VEHICLE AGE”
        ii. Void *setPrice*(float input)
            1. IF input >= 0.00
                a. *price* = input
            2. ELSE
                a. Print “INVALID VEHICLE PRICE”
        iii. Void *setHeading*(string input)
            1. *heading* = input
        iv. Void *setIsStarted*(bool input)
            1. *isStarted* = input
        v. Integer *getAge*()
            1. Return *age*

vi. Float *getPrice*()
    1. Return *price*
vii. String *getHeading*()
    1. Return *heading*
viii. Bool *getIsStarted*()
    1. Return *isStarted*
ix. Void *moveNorth*()
    1. Call *setHeading*("North")
    2. Print "Moved North"
x. Void *moveSouth*()
    1. Call *setHeading*("South")
    2. Print "Moved South"
xi. Void *moveEast*()
    1. Call *setHeading*("East")
    2. Print "Moved East"
xii. Void *moveWest*()
    1. Call *setHeading*("West")
    2. Print "Moved West"
xiii. Void *printVehicleInfo*()
    1. Print get*Age*()
    2. Print get*Price*()
    3. Print get*Heading*()
    4. Print *getIsStarted*()
xiv. Void *printVehicleOptions*()

2. Create Child Class From *vehicle*
    a. Name: *boat*
    b. Data:
        i. Bool *isDocked*
        ii. Bool *flagRaised*
    c. Functions:
        i. Void *setIsDocked*(bool input)
            1. *isDocked* = input
        ii. Void *setFlagRaised*(bool input)
            1. *flagRaised* = input
        iii. Bool *getIsDocked*()
            1. Return *isDocked*
        iv. Bool *getFlagRaised*()
            1. Return *flagRaised*
        v. Void *toggleFlag*()
            1. IF *getFlagRaised*() == TRUE
                a. Call *setFlagRaised*(FALSE)
                b. Print "Flag Lowered"
            2. ELSE

a. Call *setFlagRaised*(TRUE)
b. Print “Flag Raised”

vi. Void *undock*()
1. IF *getIsDocked*() == TRUE
a. Call *setIsDocked*(FALSE)
2. ELSE
a. Print “You Are Already Un-Docked”

vii. Void *dock*()
1. IF *getIsDocked*() == FALSE
a. Call *setIsDocked*(TRUE)
2. ELSE
a. Print “You Are Already Docked”

viii. Void *printBoatInfo*()
1. Call *printVehicleInfo*()
2. Print *isDocked*
3. Print *flagRaised*

ix. Void *printBoatOptions*()
1. Call *printVehicleInfo*()
2. Print *getIsDocked*()
3. Print *getFlagRaised*()

3. Create Child Class From *vehicle*
a. Name: *plane*
b. Data:
i. Bool *liftoffStatus*
ii. Integer *altitude*
c. Functions:
i. Void *setLiftoffStatus*(bool input)
1. liftoffStatus = input
ii. Void *setAltitude*(integer input)
1. IF input >= 0
a. altitude = input
2. ELSE
a. INVALID ALTITUDE
iii. Bool *getLiftoffStatus*()
1. Return *liftoffStatus*
iv. Integer *getAltitude*()
1. Return *altitude*
v. Void *takeoff*()
1. IF *isStarted*() == TRUE
a. IF *getLiftoffStatus*() == FALSE && *altitude* == 0
i. Call *setLiftoffStatus*(TRUE)
ii. Call *setAltitude*(1)
iii. Print “Plane Took Off”

iv. Print “ascended To “ *getAltitude*() “ Thousand Feet”

2. ELSE
    a. Print “Plane Not Started”

vi. Void *ascend*()

1. IF *isStarted*() == TRUE && *getLiftoffStatus*() == TRUE
    a. IF *getAltitude*() < 15
        i. *setAltitude*(*getAltitude*() + 1)
    b. ELSE
        i. Print “You Cannot Fly That High”
2. ELSE IF *isStarted*() == TRUE && *getLiftoffStatus*() == FALSE
    a. Print “You Must Take Off First”
3. ELSE
    a. Print “You Must Start The Plane First”

vii. Void *descend*()

1. IF *isStarted*() == TRUE && *getLiftoffStatus*() == TRUE
    a. IF *getAltitude*() > 2
        i. *setAltitude*(*getAltitude*() - 1)
    b. ELSE
        i. Print “You Cannot Fly Lower, Please Land Instead”
2. ELSE IF *isStarted*() == TRUE && *getLiftoffStatus*() == FALSE
    a. Print “You Must Take Off First”
3. ELSE
    a. Print “You Must Start The Plane First”

viii. Void *land*()

1. IF *getAltitude*() > 0
    a. *setLiftoffStatus*(FALSE)
    b. *setAltitude*(0)
    c. Print “Plane Landing From Current Altitude”

ix. Void *printPlaneInfo*()

1. Call *printVehicleInfo*()
2. Print *getliftoffStatus*()
3. Print *getAltitude*()

x. Void *printPlaneOptions*()

4. Create Child/Parent Class From *vehicle*
    a. Name: *landVehicle*
    b. Data:
        i. Integer *numPassengers*
        ii. Bool *isParked*
        iii. Bool *windshieldWiperStatus*
    c. Functions:
        i. Void *setNumPassengers*(integer input)
            1. IF input >= 1 && input <= 4
                a. *numPassengers* = input

2. ELSE
    a. INVALID PASSENGERS

ii. Void *setIsParked*(bool input)
1. *isParked* = input

iii. Void *setWindshieldWiperStatus*(bool input)
1. *windshieldWiperStatus* = input

iv. Integer *getNumPassengers*()
1. Return *numPassengers*

v. Bool *getIsParked*()
1. Return *isParked*

vi. Bool *getWindshieldWiperStatus*()
1. Return *windshieldWiperStatus*

vii. Void *toggleWindshieldWipers*()
1. IF *getWindShieldWiperStatus*() == TRUE
    a. *setWindshieldWiperStatus*(FALSE)
    b. Print “Wipers Off”
2. ELSE
    a. *setWindshieldWiperStatus*(TRUE)
    b. Print “Wipers On”

viii. Void *park*()
1. Call *setIsParked*(TRUE)

ix. Void *drive*()
1. Call *setIsParked*(FALSE)

x. Void *printLandVehicleInfo*()
1. Call *printVehicleInfo*()
2. Print *getNumPassengers*()
3. Print get*IsParked*()
4. Print *getWindshiledWiperStatus*()

xi. Void *printLandVehicleOptions*()

5. Create Child Class From *landVehicle*
    a. Name: *car*
    b. Data:
        i. Const String *radio*
        ii. Bool *raceCarStatus*
        iii. Bool *radioStatus*
    c. Functions:
        i. Void *setRaceCarStatus*(bool input)
            1. *raceCarStatus* = input
        ii. Void *setRadioStatus*(bool input)
            1. *radioStatus* = input
        iii. Bool *getRaceCarStatus*()
            1. Return *raceCarStatus*
        iv. Bool *getRadioStatus*()

1. Return *radioStatus*()

v. Void *toggleRadio*()
1. IF *getRadioStatus*() == TRUE
    a. *setRadioStatus*(FALSE)
2. ELSE
    a. *setRadioStatus*(TRUE)
    b. Print *radio*

vi. Void *toggleSpoiler*()
1. IF *getRaceCarStatus*() == TRUE
    a. Call *setRaceCarStatus*(FALSE)
    b. Print “Spoiler Lowered”
2. ELSE
    a. Call *setRaceCarStatus*(TRUE)
    b. Print “Spoiler Raised”

vii. Void *printCarInfo*()
1. Call *printLandVehicleInfo*()
2. Print *getRaceCarStatus*()
3. Print *getRadioStatus*()

viii. Void *printCarOptions*()

6. Create Child Class From *landVehicle*
    a. Name: *truck*
    b. Data:
        i. Bool *dieselTypeStatus*
        ii. Bool *fwdStatus*
    c. Functions:
        i. Void *setDieselTypeStatus*(bool input)
            1. *dieselTypeStatus* = input
        ii. Void *setFwdStatus*(bool input)
            1. *fedStatus* = input
        iii. Bool *getDieselTypeStatus*()
            1. Return *dieselTypeStatus*
        iv. Bool *getFwdStatus*()
            1. Return *fwdStatus*
        v. Void *toggleFWD*()
            1. IF *getFwdStatus* == TRUE
                a. Call *setFwdStatus*(FALSE)
                b. Print “Four Wheel Drive Disabled”
            2. ELSE
                a. Call *setFwdStatus*(TRUE)
                b. Print “Four Wheel Drive Enabled”
        vi. Void *exchangeCornOil*()
            1. IF *getDieselTypeStatus*(TRUE)
                a. Print “Silly Goose, You Can’t Reverse That”

2. ELSE
    a. *setDieselTypeStatus*(TRUE)
    b. “You’re Running On Diesel Now!”

vii. Void *printTruckInfo*()
1. Call *printLandVehicleInfo*()
2. Print *getDieselTypeStatus*()
3. Print *getFwdStatus*()

viii. Void *printTruckOptions*()

7. Create *Main*
    a. Create Instance of *car*
    b. Create Instance of *truck*
    c. Create Instance of *plane*
    d. Create Instance of *boat*
    e. Begin Selection Loop
    f. While Choice NOT ‘0’ (Switch Case?)
        i. Prompt User For Vehicle Choice
        ii. ‘C’ Enters *car*
            1. File Out “Start Trip”
            2. While Choice NOT ‘0’
                a. Prompt User for *car* Choice
                b. ‘I’ calls *printCarInfo*()
                    i. File Out “Info Requested”
                c. ‘O’ calls *printCarOptions*()
                    i. File Out “Options Printed”
                d. ‘R’ calls *toggleRadio*()
                    i. File Out “Radio Toggled”
                e. ‘J’ calls *toggleSpoiler*()
                    i. File Out “Spoiler Toggled”
                f. ‘U’ calls *toggleWindshieldWipers*()
                    i. File Out “Windshield Wipers Toggled”
                g. ‘P’ calls *park*()
                    i. File Out “Parked”
                h. ‘D’ calls *drive*()
                    i. File Out “Entered Drive”
                i. ‘N’ calls overridden *moveNorth*()
                    i. File Out “Drove North”
                j. ‘S’ calls overridden *moveSouth*()
                    i. File Out “Drove South”
                k. ‘E’ calls overridden *moveEast*()
                    i. File Out “Drove East”
                l. ‘W’ calls overridden *moveWest*()
                    i. File Out “Drove West”
            3. File Out “End Trip”

iii. ‘T’ Enters *truck*
    1. File Out “Start Trip”
    2. While Choice NOT ‘0’
        a. Prompt User for *truck* Choice
        b. ‘I’ calls *printTruckInfo*()
            i. File Out “Requested Info”
        c. ‘O’ calls *printTruckOptions*()
            i. File Out “Requested Options”
        d. ‘U’ calls *toggleWindshieldWipers*()
            i. File Out “Toggled Windshield Wipers”
        e. ‘F’ calls *toggleFWD*()
            i. File Out “Toggled 4WD”
        f. ‘C’ calls *exchangeCornOil*()
            i. File Out “Tried Diesel”
        g. ‘P’ calls *park*()
            i. File Out “Parked”
        h. ‘D’ calls *drive*()
            i. File Out “Entered Drive”
        i. ‘N’ calls overridden *moveNorth*()
            i. File Out “Drove North”
        j. ‘S’ calls overridden *moveSouth*()
            i. File Out “Drove South”
        k. ‘E’ calls overridden *moveEast*()
            i. File Out “Drove East”
        l. ‘W’ calls overridden *moveWest*()
            i. File Out “Drove West”
    3. File Out “End Trip”

iv. ‘P’ Enters *plane*
    1. Start Trip
    2. While Choice NOT ‘0’
        a. Prompt User for *plane* Choice
        b. ‘I’ calls *printPlaneInfo*()
            i. File Out “Requested Info”
        c. ‘O’ calls *printPlaneOptions*()
            i. File Out “Requested Options”
        d. ‘T’ calls *takeoff*()
            i. File Out “Took Off”
        e. ‘L’ calls *land*()
            i. File Out “Landed”
        f. ‘A’ calls *ascend*()
            i. File Out “ascended”
        g. ‘D’ calls *descend*()
            i. File Out “descended”

h. ‘N’ calls overridden *moveNorth*()
    i. File Out “Flew North”
i. ‘S’ calls overridden *moveSouth*()
    i. File Out “Flew South”
j. ‘E’ calls overridden *moveEast*()
    i. File Out “Flew East”
k. ‘W’ calls overridden *moveWest*()
    i. File Out “Flew West”

3. File Out “End Trip”

v. ‘B’ Enters *boat*

1. File Out “Start Trip”
2. While Choice NOT ‘0’
    a. Prompt User for *boat* Choice
    b. ‘I’ calls *printBoatInfo*()
        i. File Out “Requested Info”
    c. ‘O’ calls *printBoatOptions*()
        i. File Out “Requested Options”
    d. ‘F’ calls *toggleFlag*()
        i. File Out “Toggled Flag”
    e. ‘U’ calls *undock*()
        i. File Out “Un-Docked”
    f. ‘D’ calls *dock*()
        i. File Out “Docked
    g. ‘N’ calls overridden *moveNorth*()
        i. File Out “Sailed North”
    h. ‘S’ calls overridden *moveSouth*()
        i. File Out “Sailed South”
    i. ‘E’ calls overridden *moveEast*()
        i. File Out “Sailed East”
    j. ‘W’ calls overridden *moveWest*()
        i. File Out “Sailed West”
3. End Trip

7. Test Plan Version 2

Test Strategy	#	Description	Input	Expected Output	Actual	Pass/Fail

					Output	
File Existence	1	Does File Exist?				
Valid Vehicle	1	*setAge*()	Anything >= 0	Age = Anything		
Valid Vehicle	2	*setPrice*()	Anything >= 0.00	Age = Anything		
Valid Vehicle	3	*setHeading*()	“North”	“North”		
Valid Vehicle	4	*setHeading*()	“South”	“South”		
Valid Vehicle	5	*setHeading*()	“East”	“East”		
Valid Vehicle	6	*setHeading*()	“West”	“West”		
Valid Vehicle	7	*setIsStarted*()	TRUE	TRUE		
Valid Vehicle	8	*setIsStarted*()	FALSE	FALSE		
Valid Vehicle	9	*getAge*()	NULL	Positive Int		
Valid Vehicle	10	*getPrice*()	NULL	Positive Float		
Valid Vehicle	11	*moveNorth*()	NULL	Heading = North		
Valid Vehicle	12	*moveSouth*()	NULL	Heading = South		
Valid Vehicle	13	*moveEast*()	NULL	Heading = East		
Valid Vehicle	14	*moveWest*()	NULL	Heading = West		
Valid Vehicle	15	*printVehicleInfo*()	NULL	Valid *getAge*() Valid *getPrice*() Valid *getHeading*() Valid *getIsStarted*()		
Invalid Vehicle	1	*setAge*()	Anything < 0	“INVALID AGE”		
Invalid Vehicle	2	*setPrice*()	Anything < 0.00	“INVALID PRICE”		
Valid Boat	1	*setIsDocked*()	TRUE	TRUE		
Valid Boat	2	*setIsDocked*()	FALSE	FALSE		
Valid Boat	3	*setFlagRaised*()	TRUE	TRUE		

Valid Boat	4	*setFlagRaised*()	FALSE	FALSE		
Valid Boat	5	*getIsDocked*()	NULL	TRUE		
Valid Boat	6	*getIsDocked*()	NULL	FALSE		
Valid Boat	7	*getFlagRaised*()	NULL	TRUE		
Valid Boat	8	*getFlagRaised*()	NULL	FALSE		
Valid Boat	9	*toggleFlag*()	FALSE	TRUE		
Valid Boat	10	*toggleFlag*()	TRUE	FALSE		
Valid Boat	11	*undock*()	Docked	Undocked		
Valid Boat	12	*undock*()	Undocked	Already Undocked		
Valid Boat	13	*dock*()	Undocked	Docked		
Valid Boat	14	*dock*()	Docked	Already Docked		
Valid Boat	13	*printBoatInfo*()	NULL	*printVehicleInfo*() Valid get*IsDocked*() Valid *getFlagRaised*()		
Valid Plane	1	*setLiftoffStatus*()	TRUE	TRUE		
Valid Plane	2	*setLiftoffStatus*()	FALSE	FALSE		
Valid Plane	3	*setAltitude*()	Anything >= 0	Altitude = Anything		
Valid Plane	4	*getLiftoffStatus*()	NULL	TRUE		
Valid Plane	5	*getLiftoffStatus*()	NULL	FALSE		
Valid Plane	6	*getAltitude*()	NULL	0		
Valid Plane	7	*getAltitude*()	NULL	Positive Int		
Valid Plane	8	*takeoff*()	Landed	Taken Off		
Valid Plane	9	*takeoff*()	Taken Off	Already Taken Off		
Valid Plane	10	*ascend*()	Taken Off	Altitude + 1		
Valid Plane	11	*ascend*()	Altitude 15	Cannot Go Higher		
Valid Plane	12	*ascend*()	Landed	Must Take Off/Start		

Valid Plane	13	*descend*()	Taken Off	Altitude - 1		
Valid Plane	14	*descend*()	Altitude 1	Cannot Go Lower		
Valid Plane	15	*descend*()	Landed	Must Take Off/Start		
Valid Plane	16	*land*()	Altitude > 0	Landing		
Valid Plane	17	*land*()	Altitude = 0	Already Landed		
Valid Plane	19	*printPlaneInfo*()	NULL	*printVehicleInfo*() *getLiftoffStatus*() *getAltitude*()		
Invalid Plane	1	*setAltitude*()	Anything < 0	INVALID ALTITUDE		
Valid Land V	1	*setNumPass…*()	0 < Anything < 5	Anything Passengers		
Valid Land V	2	*setIsParked*()	TRUE	TRUE		
Valid Land V	3	*setIsParked*()	FALSE	FALSE		
Valid Land V	4	*setWindshield..*()	TRUE	TRUE		
Valdi Land V	5	*setWindshield..*()	FALSE	FALSE		
Valid Land V	6	*getNumPass…*()	NULL	0 < X < 5		
Valid Land V	7	*getWindshield..*()	NULL	TRUE		
Valid Land V	8	*getWindshield..*()	NULL	FALSE		
Valid Land V	9	*toggleWinshiel..*()	Active	Inactive		
Valid Land V	10	*toggleWinshiel..*()	Inactive	Active		
Valid Land V	12	*park*()	*isParked* = T	*isParked* = T		
Valid Land V	13	*drive*()	*isParked* = F	*isParked* = F		
Valid Land V	14	*printLandVehi…*()	NULL	*printVehicleInfo*() *getNumPass…*() *getIsParked*() *getWindshield…*()		
Invalid Land V	1	*setNumPass…*()	X < 1 || X > 4	INVALID AMOUNT		

Valid Car	1	*setRaceCar…*()	TRUE	TRUE		
Valid Car	2	*setRaceCar…*()	FALSE	FALSE		
Valid Car	3	*setRadioSt…*()	TRUE	TRUE		
Valid Car	4	*setRadioSt…*()	FALSE	FALSE		
Valid Car	5	*getRaceCar…*()	NULL	TRUE		
Valid Car	6	*getRaceCar…*()	NULL	FALSE		
Valid Car	7	*getRadioSt…*()	NULL	TRUE		
Valid Car	8	*getRadioSt…*()	NULL	FALSE		
Valid Car	9	*toggleRadio*()	On	Off		
Valid Car	10	*toggleRadio*()	Off	On		
Valid Car	11	*toggleSpoiler*()	Up	Down		
Valid Car	12	*toggleSpoiler*()	Down	Up		
Valid Car	13	*printCarInfo*()	NULL	*printLandVehicleInfo*() *getRaceCarStatus*() *getRadioStatus*()		
Valid Truck	1	*setDieselType…*()	TRUE	TRUE		
Valid Truck	2	*setDieselType…*()	FALSE	FALSE		
Valid Truck	3	*setFwdStatus*()	TRUE	TRUE		
Valid Truck	4	*setFwdStatus*()	FALSE	FALSE		
Valid Truck	5	*getDieselTyp…*()	NULL	TRUE		
Valid Truck	6	*getDieselTyp…*()	NULL	FALSE		
Valid Truck	7	*getFwdStatus*()	NULL	TRUE		
Valid Truck	8	*getFwdStatus*()	NULL	FALSE		
Valid Truck	9	*toggleFWD*()	Off	ON		
Valid Truck	10	*toggleFWD*()	ON	Off		
Valid Truck	11	*exchangeCorn…*()	Not Diesel	NOW Diesel		

Valid Truck	12	*exchangeCorn…*()	Diesel	Nope Sorry		
Valid Truck	13	*printTruckInfo*()	NULL	*printLandVehicleInfo*() *getDieselType…*() *getFwdStatus*()		

8. Code

[source.cpp]

```
//Program Name: Vehicle Smorgasbord
//Programmer Name: Arthur Aigeltinger IV
//Description: Lots of validation
//Date Created: 11/26/18

#include "vehicle.h"
#include "boat.h"
#include "plane.h"
#include "landVehicle.h"
#include "car.h"
#include "truck.h"

#include <cctype>
#include <fstream>

//Function Prototypes
void vehicleMenu();

int main()
{
	//Declare Vehicles
	car mustang;
	truck ranger;
	plane cessna;
	boat catamaran;

	//Invalid Test Info
	//mustang.setAge(-1);
	//mustang.setPrice(-1);

	//Valid Info
	mustang.setAge(1985);
	mustang.setPrice(6000.00);
	mustang.setHeading("North");

	ranger.setAge(2001);
	ranger.setPrice(7256.40);
	ranger.setHeading("West");
```

```
cessna.setAge(1990);
cessna.setPrice(14000);
cessna.setHeading("East");

catamaran.setAge(2010);
catamaran.setPrice(60000);
catamaran.setHeading("South");

//Declare and Open Log File
std::ofstream log("log.txt");

//Holder Variable
char input;

//Begin Log
log << "Program Started" << std::endl;

//VEHICLE MENU
do
{
	//Prompt
	vehicleMenu();
	std::cin >> input;
	input = toupper(input);

	switch (input)
	{
		//CAR
	case 'C':
		log << "Started Trip" << std::endl;
		mustang.start();
		log << "Car Started" << std::endl;
		std::cout << "Car Started" << std::endl;
		mustang.printCarOptions();
		do
		{
			std::cout << "Choice: ";
			std::cin >> input;
			input = toupper(input);
			switch (input)
			{
			case 'I':
				mustang.printCarInfo();
				log << "Car Info Requested" << std::endl;
				break;
			case 'O':
				mustang.printCarOptions();
				log << "Car Controls Requested" << std::endl;
				break;
			case 'R':
```

```
        mustang.toggleRadio();
        log << "Toggled Radio" << std::endl;
        break;
case 'J':
        mustang.toggleSpoiler();
        log << "Toggled Spoiler / Race Mode" << std::endl;
        break;
case 'U':
        mustang.toggleWindshieldWipers();
        log << "Toggled Wipers" << std::endl;
        break;
case 'P':
        mustang.park();
        log << "Shifted Into Park" << std::endl;
        break;
case 'D':
        mustang.drive();
        log << "Shifted Into Drive" << std::endl;
        break;
case 'N':
        if (mustang.moveNorth())
        {
                log << "Drove North" << std::endl;
        }
        else
        {
                log << "Failed to Drive North" << std::endl;
        }
        break;
case 'S':
        if (mustang.moveSouth())
        {
                log << "Drove South" << std::endl;
        }
        else
        {
                log << "Failed to Drive South" << std::endl;
        }
        break;
case 'E':
        if (mustang.moveEast())
        {
                log << "Drove East" << std::endl;
        }
        else
        {
                log << "Failed to Drive East" << std::endl;
        }
        break;
case 'W':
```

```
                                        if (mustang.moveWest())
                                        {
                                                log << "Drove West" << std::endl;
                                        }
                                        else
                                        {
                                                log << "Failed to Drive West" << std::endl;
                                        }
                                        break;
                                case 'Q':
                                        if (mustang.getIsParked() == false)
                                        {
                                                std::cout << "Cannot Exit Moving Vehicle" <<
std::endl;
                                                log << "Attempted to Exit Moving Vehicle" <<
std::endl;
                                                input = 'Z';
                                        }
                                        break;
                                default:
                                        break;
                                }

                        } while (input != 'Q');
                        std::cout << "Turning Off Ignition" << std::endl;
                        std::cout << "Exiting Car" << std::endl;
                        mustang.turnOff();
                        log << "Turned Off Ignition" << std::endl;
                        log << "Exited Car" << std::endl;
                        log << "Ended Trip" << std::endl;
                        break;

                        //TRUCK
                case 'T':
                        log << "Started Trip" << std::endl;
                        ranger.start();
                        log << "Truck Started" << std::endl;
                        std::cout << "Truck Started" << std::endl;
                        ranger.printTruckOptions();
                        do
                        {
                                std::cout << "Choice: ";
                                std::cin >> input;
                                input = toupper(input);
                                switch (input)
                                {
                                case 'I':
                                        ranger.printTruckInfo();
                                        log << "Truck Info Requested" << std::endl;
                                        break;
```

```
case 'O':
	ranger.printTruckOptions();
	log << "Truck Controls Requested" << std::endl;
	break;
case 'U':
	ranger.toggleWindshieldWipers();
	log << "Toggled Windshield Wipers" << std::endl;
	break;
case 'F':
	ranger.toggleFWD();
	log << "Toggled 4WD" << std::endl;
	break;
case 'C':
	ranger.exchangeCornOil();
	log << "Attempted to Change Diesel Status" << std::endl;
	break;
case 'P':
	ranger.park();
	log << "Shifted Into Park" << std::endl;
	break;
case 'D':
	ranger.drive();
	log << "Shifted Into Drive" << std::endl;
	break;
case 'N':
	if (ranger.moveNorth())
	{
		log << "Drove North" << std::endl;
	}
	else
	{
		log << "Failed to Drive North" << std::endl;
	}
	break;
case 'S':
	if (ranger.moveSouth())
	{
		log << "Drove South" << std::endl;
	}
	else
	{
		log << "Failed to Drive South" << std::endl;
	}
	break;
case 'E':
	if (ranger.moveEast())
	{
		log << "Drove East" << std::endl;
	}
	else
```

```
                                        {
                                                log << "Failed to Drive East" << std::endl;
                                        }
                                        break;
                                case 'W':
                                        if (ranger.moveWest())
                                        {
                                                log << "Drove West" << std::endl;
                                        }
                                        else
                                        {
                                                log << "Failed to Drive West" << std::endl;
                                        }
                                        break;
                                case 'Q':
                                        if (ranger.getIsParked() == false)
                                        {
                                                std::cout << "Cannot Exit Moving Vehicle" <<
std::endl;
                                                log << "Attempted to Exit Moving Vehicle" <<
std::endl;
                                                input = 'Z';
                                        }
                                        break;
                                default:
                                        break;
                                }

                        } while (input != 'Q');
                        std::cout << "Turning Off Ignition" << std::endl;
                        std::cout << "Exiting Truck" << std::endl;
                        ranger.turnOff();
                        log << "Turned Off Ignition" << std::endl;
                        log << "Exited Truck" << std::endl;
                        log << "Ended Trip" << std::endl;
                        break;

                        //PLANE
                case 'P':
                        log << "Started Trip" << std::endl;
                        cessna.start();
                        log << "Plane Started" << std::endl;
                        std::cout << "Plane Started" << std::endl;
                        cessna.printPlaneOptions();
                        do
                        {
                                std::cout << "Choice: ";
                                std::cin >> input;
                                input = toupper(input);
                                switch (input)
```

```
{
case 'I':
	cessna.printPlaneInfo();
	log << "Requested Plane Info" << std::endl;
	break;
case 'O':
	cessna.printPlaneOptions();
	log << "Requested Plane Controls" << std::endl;
	break;
case 'T':
	if (cessna.takeoff() == true)
	{
		log << "Took Off" << std::endl;
	}
	else
	{
		log << "Failed To Take Off" << std::endl;
	}
	break;
case 'L':
	if (cessna.land() == true)
	{
		log << "Landed" << std::endl;
	}
	else
	{
		log << "Failed To Land" << std::endl;
	}
	break;
case 'A':
	if (cessna.ascend() == true)
	{
		log << "Ascended" << std::endl;
	}
	else
	{
		log << "Failed to Ascend" << std::endl;
	}
	break;
case 'D':
	if (cessna.descend() == true)
	{
		log << "Descended" << std::endl;
	}
	else
	{
		log << "Failed to Descend" << std::endl;
	}
	break;
case 'N':
```

```
                        if (cessna.moveNorth())
                        {
                                log << "Flew North" << std::endl;
                        }
                        else
                        {
                                log << "Failed to Fly North" << std::endl;
                        }
                        break;
                case 'S':
                        if (cessna.moveSouth())
                        {
                                log << "Flew South" << std::endl;
                        }
                        else
                        {
                                log << "Failed to Fly South" << std::endl;
                        }
                        break;
                case 'E':
                        if (cessna.moveEast())
                        {
                                log << "Flew East" << std::endl;
                        }
                        else
                        {
                                log << "Failed to Fly East" << std::endl;
                        }
                        break;
                case 'W':
                        if (cessna.moveWest())
                        {
                                log << "Flew West" << std::endl;
                        }
                        else
                        {
                                log << "Failed to Fly West" << std::endl;
                        }
                        break;
                case 'Q':
                        if (cessna.getLiftoffStatus())
                        {
                                std::cout << "Cannot Exit Moving Vehicle" <<
std::endl;
                                log << "Attempted to Exit Moving Vehicle" <<
std::endl;
                                input = 'Z';
                        }
                        break;
                default:
```

```
                        break;
                }

        } while (input != 'Q');
        std::cout << "Turning Off Ignition" << std::endl;
        std::cout << "Exiting Plane" << std::endl;
        cessna.turnOff();
        log << "Turned Off Ignition" << std::endl;
        log << "Exited Plane" << std::endl;
        log << "Ended Trip" << std::endl;
        break;

        //BOAT
case 'B':
        log << "Started Trip" << std::endl;
        catamaran.start();
        log << "Boat Started" << std::endl;
        std::cout << "Boat Started" << std::endl;
        catamaran.printBoatOptions();
        do
        {
                std::cout << "Choice: ";
                std::cin >> input;
                input = toupper(input);
                switch (input)
                {
                case 'I':
                        catamaran.printBoatInfo();
                        log << "Requested Boat Info" << std::endl;
                        break;
                case 'O':
                        catamaran.printBoatOptions();
                        log << "Requested Boat Controls" << std::endl;
                        break;
                case 'F':
                        catamaran.toggleFlag();
                        log << "Toggled Flag" << std::endl;
                        break;
                case 'U':
                        if (catamaran.undock())
                        {
                                log << "Un-Docked" << std::endl;
                        }
                        else
                        {
                                log << "Failed to Un-Dock" << std::endl;
                        }
                        break;
                case 'D':
                        if (catamaran.dock())
```

```
		{
			log << "Docked" << std::endl;
		}
		else
		{
			log << "Failed to Dock" << std::endl;
		}
		break;
	case 'N':
		if (catamaran.moveNorth())
		{
			log << "Sailed North" << std::endl;
		}
		else
		{
			log << "Failed to Sail North" << std::endl;
		}
		break;
	case 'S':
		if (catamaran.moveSouth())
		{
			log << "Sailed South" << std::endl;
		}
		else
		{
			log << "Failed to Sail South" << std::endl;
		}
		break;
	case 'E':
		if (catamaran.moveEast())
		{
			log << "Sailed East" << std::endl;
		}
		else
		{
			log << "Failed to Sail East" << std::endl;
		}
		break;
	case 'W':
		if (catamaran.moveWest())
		{
			log << "Sailed West" << std::endl;
		}
		else
		{
			log << "Failed to Sail West" << std::endl;
		}
		break;
	case 'Q':
		if (catamaran.getIsDocked() == false)
```

```
                                        {
                                                std::cout << "Cannot Exit Un-Docked Vehicle" <<
std::endl;
                                                log << "Attempted to Exit Un-Docked Vehicle" <<
std::endl;
                                                input = 'Z';
                                        }
                                        break;
                                default:
                                        break;
                                }

                        } while (input != 'Q');
                        std::cout << "Turning Off Ignition" << std::endl;
                        std::cout << "Exiting Boat" << std::endl;
                        catamaran.turnOff();
                        log << "Turned Off Ignition" << std::endl;
                        log << "Exited Boat" << std::endl;
                        log << "Ended Trip" << std::endl;
                        break;
                case 'E':
                        break;
                default:
                        std::cout << "Invalid Option" << std::endl;
                        log << "Attempted Entering Unknown Option" << std::endl;
                        break;
                }
        } while (input != 'E');

        log << "Exiting Program" << std::endl;

        system("pause");
        return 0;
}

void vehicleMenu()
{
        std::cout << "" << std::endl;
        std::cout << "Please Select A Vehicle" << std::endl;
        std::cout << "----------------------- " << std::endl;
        std::cout << "C: Car   - Ford Mustang" << std::endl;
        std::cout << "T: Truck - Ford Ranger" << std::endl;
        std::cout << "P: Plane - Cessna 150" << std::endl;
        std::cout << "B: Boat  - Catamaran" << std::endl;
        std::cout << "E: Exit" << std::endl;
        std::cout << "Choice: ";
}
```

[vehicle.h]

```
/*PROGRAM: LAB 5
PROGRAMMER: SHERRY ROBBINS
DESCRIPTION:Create a program that use parent-child classes & inheritance
DATE CREATED:10/15/18
*/

///UPDATED INFORMATION
//Class: Vehicle
//Edited By: Arthur Aigeltinger IV
//Description: ^^^
//Date Modified: 11/26/18

//Major Changes
/*
Changed "Vehicle" to "vehicle" for class consistency
Included "string" class to introduce "heading"
*/
#pragma once

#include <iostream>
#include <string>

#ifndef VEHICLE_H
#define VEHICLE_H

class vehicle
{
public:
	//Default Constructor
	vehicle();

	//Setter Commands
	void setAge(int a);
	void setPrice(float p);
	void setHeading(std::string input);
	void setIsStarted(bool input);

	//Getter Commands
	int getAge();
	float getPrice();
	std::string getHeading();
	bool getIsStarted();

	//Other/Control Commands
	void start();
	void turnOff();
	void moveNorth();
	void moveSouth();
```

```
        void moveEast();
        void moveWest();
        void printVehicleInfo();
        void printVehicleOptions();

private:
        int age;
        float price;
        std::string heading;
        bool isStarted;
};
#endif
```

[vehicle.cpp]

```
/*PROGRAM: LAB 5
PROGRAMMER: SHERRY ROBBINS
DESCRIPTION:Create a program that use parent-child classes & inheritance
DATE CREATED:10/15/18
*/
#include "vehicle.h"

//Default Constructor
vehicle::vehicle()
{
        age = 0;
        price = 0.0;
        heading = "North";
        isStarted = false;
}

//Setter Commands
void vehicle::setAge(int a)
{
        if (a >= 0)
        {
                age = a;
        }
        else
        {
                std::cout << "INVALID VEHICLE AGE" << std::endl;
        }
}

void vehicle::setPrice(float p)
{
        if (p >= 0.00)
        {
                price = p;
        }
```

```
	else
	{
		std::cout << "INVALID VEHICLE PRICE" << std::endl;
	}
}

void vehicle::setHeading(std::string input)
{
	heading = input;
}

void vehicle::setIsStarted(bool input)
{
	isStarted = input;
}


//Getter Commands
int vehicle::getAge()
{
	return age;
}

float vehicle::getPrice()
{
	return price;
}

std::string vehicle::getHeading()
{
	return heading;
}

bool vehicle::getIsStarted()
{
	return isStarted;
}

void vehicle::start()
{
	setIsStarted(true);
}

void vehicle::turnOff()
{
	setIsStarted(false);
}

//Other Commands
void vehicle::moveNorth()
```

```
{
	setHeading("North");
	std::cout << "Moved North" << std::endl;
}

void vehicle::moveSouth()
{
	setHeading("South");
	std::cout << "Moved South" << std::endl;
}

void vehicle::moveEast()
{
	setHeading("East");
	std::cout << "Moved East" << std::endl;
}

void vehicle::moveWest()
{
	setHeading("West");
	std::cout << "Moved West" << std::endl;
}

void vehicle::printVehicleInfo()
{
	std::cout << std::endl;
	std::cout << "All Vehicle Info" << std::endl;
	std::cout << "----------------------" << std::endl;
	std::cout << "Model Year: " << getAge() << std::endl;
	std::cout << "Price:  " << getPrice() << std::endl;
	std::cout << "Current Heading: " << getHeading() << std::endl;
	std::cout << "Vehicle Running: ";

	if (getIsStarted() == true)
	{
		std::cout << "Running" << std::endl;
	}
	else
	{
		std::cout << "Not Running" << std::endl;
	}
}

//UNECCESARY
void vehicle::printVehicleOptions()
{
}
```

[boat.h]

```
#pragma once
#include "vehicle.h"

#ifndef BOAT_H
#define BOAT_H

class boat : public vehicle
{
public:
	//Default Constructor
	boat();

	//Setter Commands
	void setIsDocked(bool input);
	void setFlagRaised(bool input);

	//Getter Commands
	bool getIsDocked();
	bool getFlagRasied();

	//Other Commands
	void toggleFlag();
	bool undock();
	bool dock();
	void printBoatInfo();
	void printBoatOptions();
	bool moveNorth();
	bool moveSouth();
	bool moveEast();
	bool moveWest();

private:
	bool isDocked;
	bool flagRasied;
};
#endif
```

[boat.cpp]

```
#include "boat.h"

boat::boat()
{
	isDocked = true;
	flagRasied = false;
}

//Setter Commands
```

```
void boat::setIsDocked(bool input)
{
	isDocked = input;
}

void boat::setFlagRaised(bool input)
{
	flagRasied = input;
}

//Getter Commands
bool boat::getIsDocked()
{
	return isDocked;
}

bool boat::getFlagRasied()
{
	return flagRasied;
}

//Other Commands
void boat::toggleFlag()
{
	if (getFlagRasied() == true)
	{
		setFlagRaised(false);
		std::cout << "Flag Lowered" << std::endl;
	}
	else
	{
		setFlagRaised(true);
		std::cout << "Flag Raised" << std::endl;
	}
}

bool boat::undock()
{
	if (getIsDocked() == true)
	{
		std::cout << "Un-Docked" << std::endl;
		setIsDocked(false);
		return true;
	}
	else
	{
		std::cout << "You Are Already Un-Docked" << std::endl;
		return false;
	}
}
```

```
bool boat::dock()
{
	if (getIsDocked() == false)
	{
		std::cout << "Docked" << std::endl;
		setIsDocked(true);
		return true;
	}
	else
	{
		std::cout << "You Are Already Docked" << std::endl;
		return false;
	}
}

void boat::printBoatInfo()
{
	printVehicleInfo();
	std::cout << "Docked Status: ";

	if (getIsDocked() == true)
	{
		std::cout << "Docked" << std::endl;
	}
	else
	{
		std::cout << "Un-Docked" << std::endl;
	}

	std::cout << "Flag: ";

	if (getFlagRasied() == true)
	{
		std::cout << "Rasied" << std::endl;
	}
	else
	{
		std::cout << "Lowered" << std::endl;
	}
}

void boat::printBoatOptions()
{
	std::cout << "Boat Controls" << std::endl;
	std::cout << "---------------------------------" << std::endl;
	std::cout << "(I) Print Boat Information" << std::endl;
	std::cout << "(O) Print Boat Options / Controls" << std::endl;
	std::cout << "(F) Toggle Flag" << std::endl;
	std::cout << "(U) Un-Dock" << std::endl;
```

```
	std::cout << "(D) Dock" << std::endl;
	std::cout << "(N) Sail North" << std::endl;
	std::cout << "(S) Sail South" << std::endl;
	std::cout << "(E) Sail East" << std::endl;
	std::cout << "(W) Sail West" << std::endl;
	std::cout << "(Q) Exit Vehicle" << std::endl;
}

bool boat::moveNorth()
{
	if (getIsStarted() == false)
	{
		std::cout << "Boat is NOT Started" << std::endl;
		return false;
	}
	else if (getIsDocked() == true)
	{
		std::cout << "Boat is still docked" << std::endl;
		return false;
	}
	else
	{
		std::cout << "Sailed North" << std::endl;
		setHeading("North");
		return true;
	}
}

bool boat::moveSouth()
{
	if (getIsStarted() == false)
	{
		std::cout << "Boat is NOT Started" << std::endl;
		return false;
	}
	else if (getIsDocked() == true)
	{
		std::cout << "Boat is still docked" << std::endl;
		return false;
	}
	else
	{
		std::cout << "Sailed South" << std::endl;
		setHeading("South");
		return true;
	}
}

bool boat::moveEast()
{
```

```
	if (getIsStarted() == false)
	{
		std::cout << "Boat is NOT Started" << std::endl;
		return false;
	}
	else if (getIsDocked() == true)
	{
		std::cout << "Boat is still docked" << std::endl;
		return false;
	}
	else
	{
		std::cout << "Sailed East" << std::endl;
		setHeading("East");
		return true;
	}
}

bool boat::moveWest()
{
	if (getIsStarted() == false)
	{
		std::cout << "Boat is NOT Started" << std::endl;
		return false;
	}
	else if (getIsDocked() == true)
	{
		std::cout << "Boat is still docked" << std::endl;
		return false;
	}
	else
	{
		std::cout << "Sailed West" << std::endl;
		setHeading("West");
		return true;
	}
}
```

[plane.h]

```
#pragma once
#include "vehicle.h"

#ifndef PLANE_H
#define PLANE_H

class plane : public vehicle
{
public:
	//Default Constructor
	plane();
```

```
        //Setter Commands
        void setLiftoffStatus(bool input);
        void setAltitude(int input);

        //Getter Commands
        bool getLiftoffStatus();
        int getAltitude();

        //Other Commands
        bool takeoff();
        bool ascend();
        bool descend();
        bool land();
        void printPlaneInfo();
        void printPlaneOptions();
        bool moveNorth();
        bool moveSouth();
        bool moveEast();
        bool moveWest();


private:
        bool liftoffStatus;
        int altitude;
};
#endif
```

[plane.cpp]

```
#include "plane.h"

plane::plane()
{
        liftoffStatus = false;
        altitude = 0;
}

//Setter Commands
void plane::setLiftoffStatus(bool input)
{
        liftoffStatus = input;
}

void plane::setAltitude(int input)
{
        if (input >= 0)
        {
                altitude = input;
        }
        else
```

```
	{
		std::cout << "INVALID ALTITUDE" << std::endl;
	}
}

//Getter Commands
bool plane::getLiftoffStatus()
{
	return liftoffStatus;
}

int plane::getAltitude()
{
	return altitude;
}

bool plane::takeoff()
{
	if (getIsStarted() == true)
	{
		if (getLiftoffStatus() == false && altitude == 0)
		{
			setLiftoffStatus(true);
			setAltitude(1);
			std::cout << "Plane Took Off" << std::endl;
			std::cout << "Ascended To " << getAltitude() << " Thousand Feet" <<
std::endl;
			return true;
		}
		else
		{
			std::cout << "Plane Already Taken Off" << std::endl;
			return false;
		}
	}
	else
	{
		std::cout << "Plane Not Started" << std::endl;
		return false;
	}
}

bool plane::ascend()
{
	if (getIsStarted() == true && getLiftoffStatus() == true)
	{
		if (getAltitude() < 15)
		{
			setAltitude(getAltitude() + 1);
			std::cout << "Ascended by 1 Thousand Feet" << std::endl;
```

```
                        return true;
                }
                else
                {
                        std::cout << "You Cannot Fly That High" << std::endl;
                        return false;
                }
        }
        else if (getIsStarted() == true && getLiftoffStatus() == false)
        {
                std::cout << "You Must Take Off First" << std::endl;
                return false;
        }
        else
        {
                std::cout << "You Must Start The Plane First" << std::endl;
                return false;
        }
}

bool plane::descend()
{
        if (getIsStarted() == true && getLiftoffStatus() == true)
        {
                if (getAltitude() > 1)
                {
                        setAltitude(getAltitude() - 1);
                        std::cout << "Descended by 1 Thousand Feet" << std::endl;
                        return true;
                }
                else
                {
                        std::cout << "You Cannot Fly Lower, Please Land Instead" << std::endl;
                        return false;
                }
        }
        else if (getIsStarted() == true && getLiftoffStatus() == false)
        {
                std::cout << "You Must Take Off First" << std::endl;
                return false;
        }
        else
        {
                std::cout << "You Must Start The Plane First" << std::endl;
                return false;
        }
}

bool plane::land()
{
```

```
	if (getAltitude() > 0)
	{
		setLiftoffStatus(false);
		setAltitude(0);
		std::cout << "Plane Landing From Current Altitude" << std::endl;
		return true;
	}
	else
	{
		std::cout << "Already Landed" << std::endl;
		return false;
	}
}

void plane::printPlaneInfo()
{
	printVehicleInfo();
	std::cout << "Flight Status: ";

	if (getLiftoffStatus())
	{
		std::cout << "In Flight" << std::endl;
		std::cout << "Altitude: " << getAltitude() << " Thousand Feet" << std::endl;
	}
	else
	{
		std::cout << "Landed" << std::endl;
		std::cout << "Altitude : 0 Feet" << std::endl;
	}
}

void plane::printPlaneOptions()
{
	std::cout << "Plane Controls" << std::endl;
	std::cout << "----------------------------------" << std::endl;
	std::cout << "(I) Print Plane Information" << std::endl;
	std::cout << "(O) Print Plane Options / Controls" << std::endl;
	std::cout << "(T) Take Off" << std::endl;
	std::cout << "(L) Land" << std::endl;
	std::cout << "(A) Ascend" << std::endl;
	std::cout << "(D) Descend" << std::endl;
	std::cout << "(N) Fly North" << std::endl;
	std::cout << "(S) Fly South" << std::endl;
	std::cout << "(E) Fly East" << std::endl;
	std::cout << "(W) Fly West" << std::endl;
	std::cout << "(Q) Exit Vehicle" << std::endl;
}

bool plane::moveNorth()
{
```

```
	if (getIsStarted() == false)
	{
		std::cout << "Plane is NOT Started" << std::endl;
		return false;
	}
	else if (getLiftoffStatus() == false)
	{
		std::cout << "Plane is still landed" << std::endl;
		return false;
	}
	else
	{
		std::cout << "Flew North" << std::endl;
		setHeading("North");
		return true;
	}
}

bool plane::moveSouth()
{
	if (getIsStarted() == false)
	{
		std::cout << "Plane is NOT Started" << std::endl;
		return false;
	}
	else if (getLiftoffStatus() == false)
	{
		std::cout << "Plane is still landed" << std::endl;
		return false;
	}
	else
	{
		std::cout << "Flew South" << std::endl;
		setHeading("South");
		return true;
	}
}

bool plane::moveEast()
{
	if (getIsStarted() == false)
	{
		std::cout << "Plane is NOT Started" << std::endl;
		return false;
	}
	else if (getLiftoffStatus() == false)
	{
		std::cout << "Plane is still landed" << std::endl;
		return false;
	}
```

```
	else
	{
		std::cout << "Flew East" << std::endl;
		setHeading("East");
		return true;
	}
}

bool plane::moveWest()
{
	if (getIsStarted() == false)
	{
		std::cout << "Plane is NOT Started" << std::endl;
		return false;
	}
	else if (getLiftoffStatus() == false)
	{
		std::cout << "Plane is still landed" << std::endl;
		return false;
	}
	else
	{
		std::cout << "Flew West" << std::endl;
		setHeading("West");
		return true;
	}
}
```

[landVehicle.h]

```
#pragma once
#include "vehicle.h"

#ifndef LANDVEHICLE_H
#define LANDVEHICLE_H

class landVehicle : public vehicle
{
public:
	//Default Constructor
	landVehicle();

	//Setter Commands
	void setNumPassengers(int input);
	void setIsParked(bool input);
	void setWindshieldWiperStatus(bool input);

	//Getter Commands
	int getNumPassengers();
	bool getIsParked();
	bool getWindshieldWiperStatus();
```

```
        //Other Commands
        void toggleWindshieldWipers();
        void park();
        void drive();
        void printLandVehicleInfo();
        void printLandVehicleOptions();
        bool moveNorth();
        bool moveSouth();
        bool moveEast();
        bool moveWest();

private:
        int numPassengers;
        bool isParked;
        bool windshieldWiperStatus;
};
#endif
```

[landVehicle.cpp]

```
#include "landVehicle.h"

//Default Constructor
landVehicle::landVehicle()
{
        numPassengers = 1;
        isParked = true;
        windshieldWiperStatus = false;
}

//Setter Commands
void landVehicle::setNumPassengers(int input)
{
        if (input >= 1 && input <= 4)
        {
                numPassengers = input;
        }
        else
        {
                std::cout << "INVALID NUMBER OF PASSENGERS" << std::endl;
        }
}

void landVehicle::setIsParked(bool input)
{
        isParked = input;
}

void landVehicle::setWindshieldWiperStatus(bool input)
{
        windshieldWiperStatus = input;
```

```
}

//Getter Commands
int landVehicle::getNumPassengers()
{
	return numPassengers;
}

bool landVehicle::getIsParked()
{
	return isParked;
}

bool landVehicle::getWindshieldWiperStatus()
{
	return windshieldWiperStatus;
}


//Other Commands
void landVehicle::toggleWindshieldWipers()
{
	if (getWindshieldWiperStatus() == true)
	{
		setWindshieldWiperStatus(false);
		std::cout << "Wipers Turned Off" << std::endl;
	}
	else
	{
		setWindshieldWiperStatus(true);
		std::cout << "Wipers Turned On" << std::endl;
	}
}

void landVehicle::park()
{
	std::cout << "Shifted Into Park" << std::endl;
	setIsParked(true);
}

void landVehicle::drive()
{
	std::cout << "Shifted Into Drive" << std::endl;
	setIsParked(false);
}

void landVehicle::printLandVehicleInfo()
{
	printVehicleInfo();
	std::cout << "Passengers: " << getNumPassengers() << std::endl;
```

```
	std::cout << "Gear: ";

	if (getIsParked() == true)
	{
		std::cout << "Parked" << std::endl;
	}
	else
	{
		std::cout << "Drive" << std::endl;
	}
}

void landVehicle::printLandVehicleOptions()
{

}

bool landVehicle::moveNorth()
{
	if (getIsStarted() == false)
	{
		std::cout << "Vehicle is NOT Started" << std::endl;
		return false;
	}
	else if (getIsParked() == true)
	{
		std::cout << "Vehicle is still in PARK" << std::endl;
		return false;
	}
	else
	{
		std::cout << "Drove North" << std::endl;
		setHeading("North");
		return true;
	}
}

bool landVehicle::moveSouth()
{
	if (getIsStarted() == false)
	{
		std::cout << "Vehicle is NOT Started" << std::endl;
		return false;
	}
	else if (getIsParked() == true)
	{
		std::cout << "Vehicle is still in PARK" << std::endl;
		return false;
	}
	else
```

```
	{
		std::cout << "Drove South" << std::endl;
		setHeading("South");
		return true;
	}
}

bool landVehicle::moveEast()
{
	if (getIsStarted() == false)
	{
		std::cout << "Vehicle is NOT Started" << std::endl;
		return false;
	}
	else if (getIsParked() == true)
	{
		std::cout << "Vehicle is still in PARK" << std::endl;
		return false;
	}
	else
	{
		std::cout << "Drove East" << std::endl;
		setHeading("East");
		return true;
	}
}

bool landVehicle::moveWest()
{
	if (getIsStarted() == false)
	{
		std::cout << "Vehicle is NOT Started" << std::endl;
		return false;
	}
	else if (getIsParked() == true)
	{
		std::cout << "Vehicle is still in PARK" << std::endl;
		return false;
	}
	else
	{
		std::cout << "Drove West" << std::endl;
		setHeading("West");
		return true;
	}
}
```

[car.h]

```
#pragma once
#include "landVehicle.h"
```

```
/*PROGRAM: LAB 5
PROGRAMMER: SHERRY ROBBINS
DESCRIPTION:Create a program that use parent-child classes & inheritance
DATE CREATED:10/16/18
*/

#ifndef CAR_H
#define CAR_H

class car : public landVehicle
{
public:
	//Default Constructor
	car();

	//Setter Commands
	void setRaceCarStatus(bool s);
	void setRadioStatus(bool input);

	//Getter Commands
	bool getRaceCarStatus();
	bool getRadioStatus();

	//Other Commands
	void toggleRadio();
	void toggleSpoiler();
	void printCarInfo();
	void printCarOptions();

private:
	bool RaceCarStatus;
	bool radioStatus;
};
#endif
```

[car.cpp]

```
#include "car.h"
/*PROGRAM: LAB 5
PROGRAMMER: SHERRY ROBBINS
DESCRIPTION:Create a program that use parent-child classes & inheritance
DATE CREATED:10/16/18
*/

//Default Constructor
car::car()
{
	RaceCarStatus = false;
	radioStatus = false;
}
```

```
//Setter Commands
void car::setRaceCarStatus(bool s)
{
	RaceCarStatus = s;
}

void car::setRadioStatus(bool input)
{
	radioStatus = input;
}

//Getter Commands
bool car::getRaceCarStatus()
{
	return RaceCarStatus;
}

bool car::getRadioStatus()
{
	return radioStatus;
}


//Other Commands
void car::toggleRadio()
{
	if (getRadioStatus() == true)
	{
		setRadioStatus(false);
		std::cout << "Radio Turned Off" << std::endl;
	}
	else
	{
		setRadioStatus(true);
		std::cout << "Radio Turned On" << std::endl;
		std::cout << "NEVER GONNA GIVE YOU UP!" << std::endl;
		std::cout << "NEVER GONNA LET YOU DOWN!" << std::endl;
		std::cout << "NEVER GONNA TURN AROUND..." << std::endl;
		std::cout << "AND HURT YOU......." << std::endl;
	}
}

void car::toggleSpoiler()
{
	if (getRaceCarStatus() == true)
	{
		setRaceCarStatus(false);
		std::cout << "Spoiler Lowered" << std::endl;
	}
	else
```

```
	{
		setRaceCarStatus(true);
		std::cout << "Spoiler Raised" << std::endl;
	}
}

void car::printCarInfo()
{
	printLandVehicleInfo();
	std::cout << "Spoiler: ";

	if (getRaceCarStatus() == true)
	{
		std::cout << "Raised" << std::endl;
	}
	else
	{
		std::cout << "Lowered" << std::endl;
	}
}

void car::printCarOptions()
{
	std::cout << "Car Controls" << std::endl;
	std::cout << "--------------------------------" << std::endl;
	std::cout << "(I) Print Car Information" << std::endl;
	std::cout << "(O) Print Car Options / Controls" << std::endl;
	std::cout << "(R) Toggle Radio" << std::endl;
	std::cout << "(J) Toggle Spoiler / JDM Mode" << std::endl;
	std::cout << "(U) Toggle Windshield Wipers" << std::endl;
	std::cout << "(P) Shift Into Park" << std::endl;
	std::cout << "(D) Shift Into Drive" << std::endl;
	std::cout << "(N) Drive North" << std::endl;
	std::cout << "(S) Drive South" << std::endl;
	std::cout << "(E) Drive East" << std::endl;
	std::cout << "(W) Drive West" << std::endl;
	std::cout << "(Q) Exit Vehicle" << std::endl;
}
```

[truck.h]

```
#pragma once
#include "landVehicle.h"
/*PROGRAM: LAB 5
PROGRAMMER: SHERRY ROBBINS
DESCRIPTION:Create a program that use parent-child classes & inheritance
DATE CREATED:10/16/18
*/

#ifndef TRUCK_H
#define TRUCK_H
```

```
class truck : public landVehicle
{
public:
	//Default Constructor
	truck();

	//Setter Commands
	void setDieselStatus(bool s);
	void setFwdStatus(bool input);

	//Getter Commands
	bool getDieselTypeStatus();
	bool getFwdStatus();

	//Other Commands
	void toggleFWD();
	void exchangeCornOil();
	void printTruckInfo();
	void printTruckOptions();

private:
	bool dieselTypeStatus;
	bool fwdStatus;
};
#endif
```

[truck.cpp]

```
/*PROGRAM: LAB 5
PROGRAMMER: SHERRY ROBBINS
DESCRIPTION:Create a program that use parent-child classes & inheritance
DATE CREATED:10/16/18
*/
#include "truck.h"

//Default Constructor
truck::truck()
{
	dieselTypeStatus = false;
	fwdStatus = false;
}

//Setter Commands
void truck::setDieselStatus(bool s)
{
	dieselTypeStatus = s;
}

void truck::setFwdStatus(bool input)
{
```

```
	fwdStatus = input;
}

//Getter Commands
bool truck::getDieselTypeStatus()
{
	return dieselTypeStatus;
}

bool truck::getFwdStatus()
{
	return fwdStatus;
}


//Other Commands
void truck::toggleFWD()
{
	if (getFwdStatus() == true)
	{
		setFwdStatus(false);
		std::cout << "Four Wheel Drive Disabled" << std::endl;
	}
	else
	{
		setFwdStatus(true);
		std::cout << "Four Wheel Drive Enabled" << std::endl;
	}
}

void truck::exchangeCornOil()
{
	if (getDieselTypeStatus() == true)
	{
		std::cout << "Silly Goose, You Can't Reverse That!" << std::endl;
	}
	else
	{
		setDieselStatus(true);
		std::cout << "You're Running On Diesel Now. Go Roll Some Coal!" << std::endl;
		std::cout << "just kidding, don't do that, its a richard move" << std::endl;
	}
}

void truck::printTruckInfo()
{
	printLandVehicleInfo();
	std::cout << "Fuel Type: ";

	if (getDieselTypeStatus() == true)
```

```
	{
		std::cout << "Yee Haw Diesel" << std::endl;
	}
	else
	{
		std::cout << "Gasoline" << std::endl;
	}
	std::cout << "4WD Status: ";

	if (getFwdStatus())
	{
		std::cout << "Active" << std::endl;
	}
	else
	{
		std::cout << "Inactive" << std::endl;
	}
}

void truck::printTruckOptions()
{
	std::cout << "Truck Controls" << std::endl;
	std::cout << "----------------------------------" << std::endl;
	std::cout << "(I) Print Truck Information" << std::endl;
	std::cout << "(O) Print Truck Options / Controls" << std::endl;
	std::cout << "(C) Exchange Corn Oil" << std::endl;
	std::cout << "(F) Toggle 4WD" << std::endl;
	std::cout << "(U) Toggle Windshield Wipers" << std::endl;
	std::cout << "(P) Shift Into Park" << std::endl;
	std::cout << "(D) Shift Into Drive" << std::endl;
	std::cout << "(N) Drive North" << std::endl;
	std::cout << "(S) Drive South" << std::endl;
	std::cout << "(E) Drive East" << std::endl;
	std::cout << "(W) Drive West" << std::endl;
	std::cout << "(Q) Exit Vehicle" << std::endl;
}
```

9. Updated Algorithm

1. Create Parent Class
    a. Name: *vehicle*
    b. Data:
        i. Integer *age*
        ii. Float *price*
        iii. String *heading*
        iv. Bool *isStarted*
    c. Functions:
        i. Void *setAge*(integer input)

1. IF input >= 0
    a. *age* = input
2. ELSE
    a. Print “INVALID VEHICLE AGE”

ii. Void *setPrice*(float input)
1. IF input >= 0.00
    a. *price* = input
2. ELSE
    a. Print “INVALID VEHICLE PRICE”

iii. Void *setHeading*(string input)
1. *heading* = input

iv. Void *setIsStarted*(bool input)
1. *isStarted* = input

v. Integer *getAge*()
1. Return *age*

vi. Float *getPrice*()
1. Return *price*

vii. String *getHeading*()
1. Return *heading*

viii. Bool *getIsStarted*()
1. Return *isStarted*

ix. Void *start*()
1. Call *setIsStarted*(TRUE)

x. Void *turnOff*()
1. Call *setIsStarted*(FALSE)

xi. Void *moveNorth*()
1. Call *setHeading*(“North”)
2. Print “Moved North”

xii. Void *moveSouth*()
1. Call *setHeading*(“South”)
2. Print “Moved South”

xiii. Void *moveEast*()
1. Call *setHeading*(“East”)
2. Print “Moved East”

xiv. Void *moveWest*()
1. Call *setHeading*(“West”)
2. Print “Moved West”

xv. Void *printVehicleInfo*()
1. Print get*Age*()
2. Print get*Price*()
3. Print get*Heading*()
4. Print *getIsStarted*()

xvi. Void *printVehicleOptions*()

1. Print Controls

2. Create Child Class From *vehicle*
    a. Name: *boat*
    b. Data:
        i. Bool *isDocked*
        ii. Bool *flagRaised*
    c. Functions:
        i. Void *setIsDocked*(bool input)
            1. *isDocked* = input
        ii. Void *setFlagRaised*(bool input)
            1. *flagRaised* = input
        iii. Bool *getIsDocked*()
            1. Return *isDocked*
        iv. Bool *getFlagRaised*()
            1. Return *flagRaised*
        v. Void *toggleFlag*()
            1. IF *getFlagRaised*() == TRUE
                a. Call *setFlagRaised*(FALSE)
                b. Print “Flag Lowered”
            2. ELSE
                a. Call *setFlagRaised*(TRUE)
                b. Print “Flag Raised”
        vi. Void *undock*()
            1. IF *getIsDocked*() == TRUE
                a. Call *setIsDocked*(FALSE)
            2. ELSE
                a. Print “You Are Already Un-Docked”
        vii. Void *dock*()
            1. IF *getIsDocked*() == FALSE
                a. Call *setIsDocked*(TRUE)
            2. ELSE
                a. Print “You Are Already Docked”
        viii. Void *printBoatInfo*()
            1. Call *printVehicleInfo*()
            2. Print *isDocked*
            3. Print *flagRaised*
        ix. Void *printBoatOptions*()
            ~~1. Call *printVehicleInfo*()~~
            ~~2. Print *getIsDocked*()~~
            ~~3. Print *getFlagRaised*()~~
            4. Print Controls
        x.

3. Create Child Class From *vehicle*

a. Name: *plane*
b. Data:
    i. Bool *liftoffStatus*
    ii. Integer *altitude*
c. Functions:
    i. Void *setLiftoffStatus*(bool input)
        1. *liftoffStatus* = input
    ii. Void *setAltitude*(integer input)
        1. IF input >= 0
            a. *altitude* = input
        2. ELSE
            a. INVALID ALTITUDE
    iii. Bool *getLiftoffStatus*()
        1. Return *liftoffStatus*
    iv. Integer *getAltitude*()
        1. Return *altitude*
    v. Void *takeoff*()
        1. IF *isStarted*() == TRUE
            a. IF *getLiftoffStatus*() == FALSE && *altitude* == 0
                i. Call *setLiftoffStatus*(TRUE)
                ii. Call *setAltitude*(1)
                iii. Print “Plane Took Off”
                iv. Print “Ascended To “ *getAltitude*() “ Thousand Feet”
        2. ELSE
            a. Print “Plane Not Started”
    vi. Void *ascend*()
        1. IF *isStarted*() == TRUE && *getLiftoffStatus*() == TRUE
            a. IF *getAltitude*() < 15
                i. *setAltitude*(*getAltitude*() + 1)
            b. ELSE
                i. Print “You Cannot Fly That High”
        2. ELSE IF *isStarted*() == TRUE && *getLiftoffStatus*() == FALSE
            a. Print “You Must Take Off First”
        3. ELSE
            a. Print “You Must Start The Plane First”
    vii. Void *descend*()
        1. IF *isStarted*() == TRUE && *getLiftoffStatus*() == TRUE
            a. IF *getAltitude*() > 2
                i. *setAltitude*(*getAltitude*() - 1)
            b. ELSE
                i. Print “You Cannot Fly Lower, Please Land Instead”
        2. ELSE IF *isStarted*() == TRUE && *getLiftoffStatus*() == FALSE
            a. Print “You Must Take Off First”

3. ELSE
    a. Print “You Must Start The Plane First”

viii. Void *land*()
1. IF *getAltitude*() > 0
    a. *setLiftoffStatus*(FALSE)
    b. *setAltitude*(0)
    c. Print “Plane Landing From Current Altitude”

ix. Void *printPlaneInfo*()
1. Call *printVehicleInfo*()
2. Print *getliftoffStatus*()
3. Print *getAltitude*()

x. Void *printPlaneOptions*()
1. Print Controls

4. Create Child/Parent Class From *vehicle*
    a. Name: *landVehicle*
    b. Data:
        i. Integer *numPassengers*
        ii. Bool *isParked*
        iii. Bool *windshieldWiperStatus*
    c. Functions:
        i. Void *setNumPassengers*(integer input)
            1. IF input >= 1 && input <= 4
                a. *numPassengers* = input
            2. ELSE
                a. INVALID PASSENGERS
        ii. Void *setIsParked*(bool input)
            1. *isParked* = input
        iii. Void *setWindshieldWiperStatus*(bool input)
            1. *windshieldWiperStatus* = input
        iv. Integer *getNumPassengers*()
            1. Return *numPassengers*
        v. Bool *getIsParked*()
            1. Return *isParked*
        vi. Bool *getWindshieldWiperStatus*()
            1. Return *windshieldWiperStatus*
        vii. Void *toggleWindshieldWipers*()
            1. IF *getWindShieldWiperStatus*() == TRUE
                a. *setWindshieldWiperStatus*(FALSE)
                b. Print “Wipers Off”
            2. ELSE
                a. *setWindshieldWiperStatus*(TRUE)
                b. Print “Wipers On”
        viii. Void *park*()

1. Call *setIsParked*(TRUE)

ix. Void *drive*()
1. Call *setIsParked*(FALSE)

x. Void *printLandVehicleInfo*()
1. Call *printVehicleInfo*()
2. Print *getNumPassengers*()
3. Print get*IsParked*()
4. Print *getWindshiledWiperStatus*()

xi. Void *printLandVehicleOptions*()

5. Create Child Class From *landVehicle*
   a. Name: *car*
   b. Data:
      i. Const String *radio*
      ii. Bool *raceCarStatus*
      iii. Bool *radioStatus*
   c. Functions:
      i. Void *setRaceCarStatus*(bool input)
         1. *raceCarStatus* = input
      ii. Void *setRadioStatus*(bool input)
         1. *radioStatus* = input
      iii. Bool *getRaceCarStatus*()
         1. Return *raceCarStatus*
      iv. Bool *getRadioStatus*()
         1. Return *radioStatus*()
      v. Void *toggleRadio*()
         1. IF *getRadioStatus*() == TRUE
            a. *setRadioStatus*(FALSE)
         2. ELSE
            a. *setRadioStatus*(TRUE)
            b. Print *radio*
      vi. Void *toggleSpoiler*()
         1. IF *getRaceCarStatus*() == TRUE
            a. Call *setRaceCarStatus*(FALSE)
            b. Print “Spoiler Lowered”
         2. ELSE
            a. Call *setRaceCarStatus*(TRUE)
            b. Print “Spoiler Raised”
      vii. Void *printCarInfo*()
         1. Call *printLandVehicleInfo*()
         2. Print *getRaceCarStatus*()
         3. Print *getRadioStatus*()
      viii. Void *printCarOptions*()
6. Create Child Class From *landVehicle*

a. Name: *truck*
b. Data:
    i. Bool *dieselTypeStatus*
    ii. Bool *fwdStatus*
c. Functions:
    i. Void *setDieselTypeStatus*(bool input)
        1. *dieselTypeStatus* = input
    ii. Void *setFwdStatus*(bool input)
        1. *fwdStatus* = input
    iii. Bool *getDieselTypeStatus*()
        1. Return *dieselTypeStatus*
    iv. Bool *getFwdStatus*()
        1. Return *fwdStatus*
    v. Void *toggleFWD*()
        1. IF *getFwdStatus* == TRUE
            a. Call *setFwdStatus*(FALSE)
            b. Print “Four Wheel Drive Disabled”
        2. ELSE
            a. Call *setFwdStatus*(TRUE)
            b. Print “Four Wheel Drive Enabled”
    vi. Void *exchangeCornOil*()
        1. IF *getDieselTypeStatus*(TRUE)
            a. Print “Silly Goose, You Can’t Reverse That”
        2. ELSE
            a. *setDieselTypeStatus*(TRUE)
            b. “You’re Running On Diesel Now!”
    vii. Void *printTruckInfo*()
        1. Call *printLandVehicleInfo*()
        2. Print *getDieselTypeStatus*()
        3. Print *getFwdStatus*()
    viii. Void *printTruckOptions*()
        1. Print Controls

7. Create *Main*
    a. Create Instance of *car*
    b. Create Instance of *truck*
    c. Create Instance of *plane*
    d. Create Instance of *boat*
    e. Begin Selection Loop
    f. While Choice NOT ‘E’ (Switch Case?)
        i. Prompt User For Vehicle Choice
        ii. ‘C’ Enters *car*
            1. File Out “Start Trip”
            2. While Choice NOT ‘Q’

a. Prompt User for *car* Choice
b. 'I' calls *printCarInfo*()
    i. File Out "Info Requested"
c. 'O' calls *printCarOptions*()
    i. File Out "Options Printed"
d. 'R' calls *toggleRadio*()
    i. File Out "Radio Toggled"
e. 'J' calls *toggleSpoiler*()
    i. File Out "Spoiler Toggled"
f. 'U' calls *toggleWindshieldWipers*()
    i. File Out "Windshield Wipers Toggled"
g. 'P' calls *park*()
    i. File Out "Parked"
h. 'D' calls *drive*()
    i. File Out "Entered Drive"
i. 'N' calls overridden *moveNorth*()
    i. File Out "Drove North"
j. 'S' calls overridden *moveSouth*()
    i. File Out "Drove South"
k. 'E' calls overridden *moveEast*()
    i. File Out "Drove East"
l. 'W' calls overridden *moveWest*()
    i. File Out "Drove West"

3. File Out "End Trip"

iii. 'T' Enters *truck*

1. File Out "Start Trip"
2. While Choice NOT 'Q'
    a. Prompt User for *truck* Choice
    b. 'I' calls *printTruckInfo*()
        i. File Out "Requested Info"
    c. 'O' calls *printTruckOptions*()
        i. File Out "Requested Options"
    d. 'U' calls *toggleWindshieldWipers*()
        i. File Out "Toggled Windshield Wipers"
    e. 'F' calls *toggleFWD*()
        i. File Out "Toggled 4WD"
    f. 'C' calls *exchangeCornOil*()
        i. File Out "Tried Diesel"
    g. 'P' calls *park*()
        i. File Out "Parked"
    h. 'D' calls *drive*()
        i. File Out "Entered Drive"
    i. 'N' calls overridden *moveNorth*()

i. File Out “Drove North”

j. ‘S’ calls overridden *moveSouth*()

i. File Out “Drove South”

k. ‘E’ calls overridden *moveEast*()

i. File Out “Drove East”

l. ‘W’ calls overridden *moveWest*()

i. File Out “Drove West”

3. File Out “End Trip”

iv. ‘P’ Enters *plane*

1. Start Trip
2. While Choice NOT ‘Q’
   a. Prompt User for *plane* Choice
   b. ‘I’ calls *printPlaneInfo*()
      i. File Out “Requested Info”
   c. ‘O’ calls *printPlaneOptions*()
      i. File Out “Requested Options”
   d. ‘T’ calls *takeoff*()
      i. File Out “Took Off”
   e. ‘L’ calls *land*()
      i. File Out “Landed”
   f. ‘A’ calls *ascend*()
      i. File Out “Ascended”
   g. ‘D’ calls *descend*()
      i. File Out “Descended”
   h. ‘N’ calls overridden *moveNorth*()
      i. File Out “Flew North”
   i. ‘S’ calls overridden *moveSouth*()
      i. File Out “Flew South”
   j. ‘E’ calls overridden *moveEast*()
      i. File Out “Flew East”
   k. ‘W’ calls overridden *moveWest*()
      i. File Out “Flew West”
3. File Out “End Trip”

v. ‘B’ Enters *boat*

1. File Out “Start Trip”
2. While Choice NOT ‘’
   a. Prompt User for *boat* Choice
   b. ‘I’ calls *printBoatInfo*()
      i. File Out “Requested Info”
   c. ‘O’ calls *printBoatOptions*()
      i. File Out “Requested Options”
   d. ‘F’ calls *toggleFlag*()
      i. File Out “Toggled Flag”

e. ‘U’ calls *undock*()
    i. File Out “Un-Docked”
f. ‘D’ calls *dock*()
    i. File Out “Docked
g. ‘N’ calls overridden *moveNorth*()
    i. File Out “Sailed North”
h. ‘S’ calls overridden *moveSouth*()
    i. File Out “Sailed South”
i. ‘E’ calls overridden *moveEast*()
    i. File Out “Sailed East”
j. ‘W’ calls overridden *moveWest*()
    i. File Out “Sailed West”

3. End Trip

10. Test Plan Version 3

Test Strategy	#	Description	Input	Expected Output	Actual Output	Pass/Fail
File Existence	1	Does File Exist?	log.txt	log.txt	log.txt	Pass
Valid Vehicle	1	*setAge*()	Anything >= 0	Age = Anything	Age = Anything	Pass
Valid Vehicle	2	*setPrice*()	Anything >= 0.00	Age = Anything	Age = Anything	Pass
Valid Vehicle	3	*setHeading*()	“North”	“North”	“North”	Pass
Valid Vehicle	4	*setHeading*()	“South”	“South”	“South”	Pass
Valid Vehicle	5	*setHeading*()	“East”	“East”	“East”	Pass
Valid Vehicle	6	*setHeading*()	“West”	“West”	“West”	Pass
Valid Vehicle	7	*setIsStarted*()	TRUE	TRUE	TRUE	Pass
Valid Vehicle	8	*setIsStarted*()	FALSE	FALSE	FALSE	Pass
Valid Vehicle	9	*getAge*()	NULL	Positive Int	Positive Int	Pass
Valid Vehicle	10	*getPrice*()	NULL	Positive Float	Positive Float	Pass
Valid Vehicle	11	*moveNorth*()	NULL	Heading = North	Heading = North	Pass
Valid Vehicle	12	*moveSouth*()	NULL	Heading = South	Heading = South	Pass

Valid Vehicle	13	*moveEast*()	NULL	Heading = East	Heading = East	Pass
Valid Vehicle	14	*moveWest*()	NULL	Heading = West	Heading = West	Pass
Valid Vehicle	15	*printVehicleInfo*()	NULL	Valid *getAge*() Valid *getPrice*() Valid *getHeading*() Valid *getIsStarted*()	Valid *getAge*() Valid *getPrice*() Valid *getHeading*() Valid *getIsStarted*()	Pass
Invalid Vehicle	1	*setAge*()	Anything < 0	“INVALID AGE”	“INVALID AGE”	Pass
Invalid Vehicle	2	*setPrice*()	Anything < 0.00	“INVALID PRICE”	“INVALID PRICE”	Pass
Valid Boat	1	*setIsDocked*()	TRUE	TRUE	TRUE	Pass
Valid Boat	2	*setIsDocked*()	FALSE	FALSE	FALSE	Pass
Valid Boat	3	*setFlagRaised*()	TRUE	TRUE	TRUE	Pass
Valid Boat	4	*setFlagRaised*()	FALSE	FALSE	FALSE	Pass
Valid Boat	5	*getIsDocked*()	NULL	TRUE	TRUE	Pass
Valid Boat	6	*getIsDocked*()	NULL	FALSE	FALSE	Pass
Valid Boat	7	*getFlagRaised*()	NULL	TRUE	TRUE	Pass
Valid Boat	8	*getFlagRaised*()	NULL	FALSE	FALSE	Pass
Valid Boat	9	*toggleFlag*()	FALSE	TRUE	TRUE	Pass
Valid Boat	10	*toggleFlag*()	TRUE	FALSE	FALSE	Pass
Valid Boat	11	*undock*()	Docked	Undocked	Undocked	Pass
Valid Boat	12	*undock*()	Undocked	Already Undocked	Already Undocked	Pass
Valid Boat	13	*dock*()	Undocked	Docked	Docked	Pass
Valid Boat	14	*dock*()	Docked	Already Docked	Already Docked	Pass
Valid Boat	13	*printBoatInfo*()	NULL	*printVehicleInfo*() Valid get*IsDocked*() Valid	*printVehicleInfo*() Valid get*IsDocked*() Valid	Pass

				getFlagRaised()	*getFlagRaised*()	
Valid Plane	1	*setLiftoffStatus*()	TRUE	TRUE	TRUE	Pass
Valid Plane	2	*setLiftoffStatus*()	FALSE	FALSE	FALSE	Pass
Valid Plane	3	*setAltitude*()	Anything >= 0	Altitude = Anything	Altitude = Anything	Pass
Valid Plane	4	*getLiftoffStatus*()	NULL	TRUE	TRUE	Pass
Valid Plane	5	*getLiftoffStatus*()	NULL	FALSE	FALSE	Pass
Valid Plane	6	*getAltitude*()	NULL	0	0	Pass
Valid Plane	7	*getAltitude*()	NULL	Positive Int	Positive Int	Pass
Valid Plane	8	*takeoff*()	Landed	Taken Off	Taken Off	Pass
Valid Plane	9	*takeoff*()	Taken Off	Already Taken Off	Already Taken Off	Pass
Valid Plane	10	*ascend*()	Taken Off	Altitude + 1	Altitude + 1	Pass
Valid Plane	11	*ascend*()	Altitude 15	Cannot Go Higher	Cannot Go Higher	Pass
Valid Plane	12	*ascend*()	Landed	Must Take Off/Start	Must Take Off/Start	Pass
Valid Plane	13	*descend*()	Taken Off	Altitude - 1	Altitude - 1	Pass
Valid Plane	14	*descend*()	Altitude 1	Cannot Go Lower	Cannot Go Lower	Pass
Valid Plane	15	*descend*()	Landed	Must Take Off/Start	Must Take Off/Start	Pass
Valid Plane	16	*land*()	Altitude > 0	Landing	Landing	Pass
Valid Plane	17	*land*()	Altitude = 0	Already Landed	Already Landed	Pass
Valid Plane	19	*printPlaneInfo*()	NULL	*printVehicleInfo*() *getLiftoffStatus*() *getAltitude*()	*printVehicleInfo*() *getLiftoffStatus*() *getAltitude*()	Pass
Valid Plane	20	*move…*()	Landed	*INVALID*	*INVALID*	Pass

Valid Plane	21	*move..*()	Taken Off	*Valid MOve*	*Valid MOve*	Pass
Invalid Plane	1	*setAltitude*()	Anything < 0	INVALID ALTITUDE	INVALID ALTITUDE	Pass
Valid Land V	1	*setNumPass…*()	0 < Anything < 5	Anything Passengers	Anything Passengers	Pass
Valid Land V	2	*setIsParked*()	TRUE	TRUE	TRUE	Pass
Valid Land V	3	*setIsParked*()	FALSE	FALSE	FALSE	Pass
Valid Land V	4	*setWindshield..*()	TRUE	TRUE	TRUE	Pass
Valdi Land V	5	*setWindshield..*()	FALSE	FALSE	FALSE	Pass
Valid Land V	6	*getNumPass…*()	NULL	0 < X < 5	0 < X < 5	Pass
Valid Land V	7	*getWindshield..*()	NULL	TRUE	TRUE	Pass
Valid Land V	8	*getWindshield..*()	NULL	FALSE	FALSE	Pass
Valid Land V	9	*toggleWinshiel..*()	Active	Inactive	Inactive	Pass
Valid Land V	10	*toggleWinshiel..*()	Inactive	Active	Active	Pass
Valid Land V	12	*park*()	*isParked* = T	*isParked* = T	*isParked* = T	Pass
Valid Land V	13	*drive*()	*isParked* = F	*isParked* = F	*isParked* = F	Pass
Valid Land V	14	*printLandVehi…*()	NULL	*printVehicleInfo*() *getNumPass…*() *getIsParked*() *getWindshield…*( )	*printVehicleInfo*() *getNumPass…*() *getIsParked*() *getWindshield…*( )	Pass
Invalid Land V	1	*setNumPass…*()	X < 1 || X > 4	INVALID AMOUNT	INVALID AMOUNT	Pass
						Pass
Valid Car	1	*setRaceCar…*()	TRUE	TRUE	TRUE	Pass
Valid Car	2	*setRaceCar…*()	FALSE	FALSE	FALSE	Pass
Valid Car	3	*setRadioSt…*()	TRUE	TRUE	TRUE	Pass
Valid Car	4	*setRadioSt…*()	FALSE	FALSE	FALSE	Pass

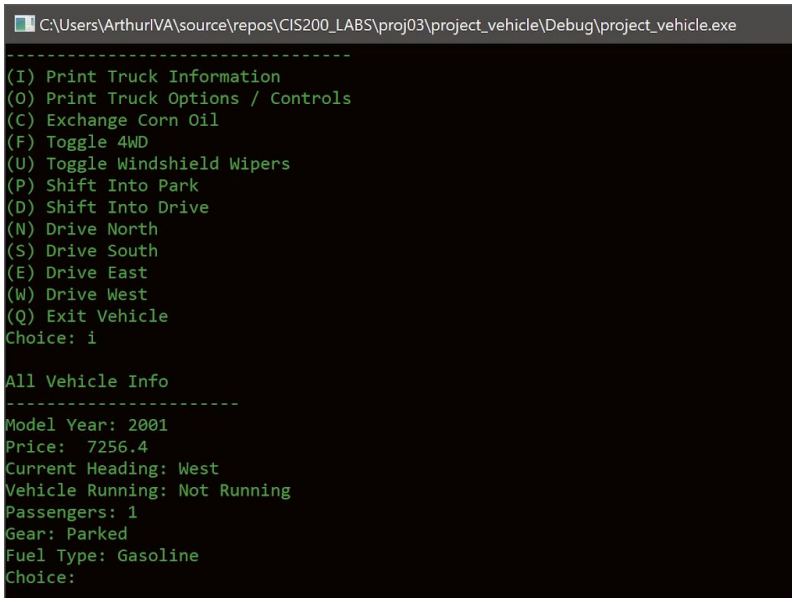
Valid Car	5	*getRaceCar…*()	NULL	TRUE	TRUE	Pass
Valid Car	6	*getRaceCar…*()	NULL	FALSE	FALSE	Pass
Valid Car	7	*getRadioSt…*()	NULL	TRUE	TRUE	Pass
Valid Car	8	*getRadioSt…*()	NULL	FALSE	FALSE	Pass
Valid Car	9	*toggleRadio*()	On	Off	Off	Pass
Valid Car	10	*toggleRadio*()	Off	On	On	Pass
Valid Car	11	*toggleSpoiler*()	Up	Down	Down	Pass
Valid Car	12	*toggleSpoiler*()	Down	Up	Up	Pass
Valid Car	13	*printCarInfo*()	NULL	*printLandVehicleInfo*() *getRaceCarStatus*() *getRadioStatus*()	*printLandVehicleInfo*() *getRaceCarStatus*() *getRadioStatus*()	Pass
Valid Truck	1	*setDieselType…*()	TRUE	TRUE	TRUE	Pass
Valid Truck	2	*setDieselType…*()	FALSE	FALSE	FALSE	Pass
Valid Truck	3	*setFwdStatus*()	TRUE	TRUE	TRUE	Pass
Valid Truck	4	*setFwdStatus*()	FALSE	FALSE	FALSE	Pass
Valid Truck	5	*getDieselTyp…*()	NULL	TRUE	TRUE	Pass
Valid Truck	6	*getDieselTyp…*()	NULL	FALSE	FALSE	Pass
Valid Truck	7	*getFwdStatus*()	NULL	TRUE	TRUE	Pass
Valid Truck	8	*getFwdStatus*()	NULL	FALSE	FALSE	Pass
Valid Truck	9	*toggleFWD*()	Off	ON	ON	Pass
Valid Truck	10	*toggleFWD*()	ON	Off	Off	Pass
Valid Truck	11	*exchangeCorn…*()	Not Diesel	NOW Diesel	NOW Diesel	Pass
Valid Truck	12	*exchangeCorn…*()	Diesel	Nope Sorry	Nope Sorry	Pass
Valid Truck	13	*printTruckInfo*()	NULL	*printLandVehicleInfo*()	*printLandVehicleInfo*()	Pass

				getDieselType…() *getFwdStatus*()	*getDieselType…()* *getFwdStatus*()	
Path	1	Test Path of All Options in Every Vehicle	Map Provided	log.txt	*log.txt*	Pass

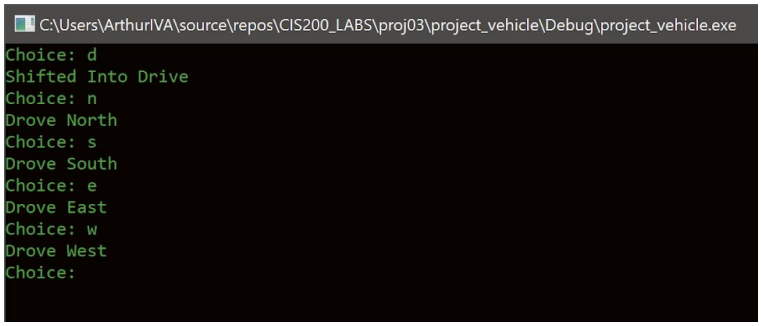
11. Screenshots

Valid Vehicle 1, 2, 3, 7, 8, 9, 10, 15.

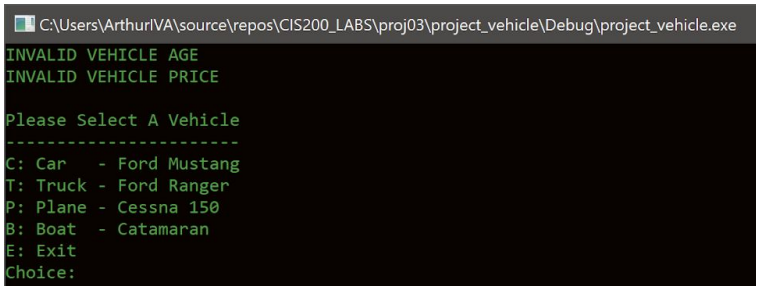
Valid Vehicle 6, 8.



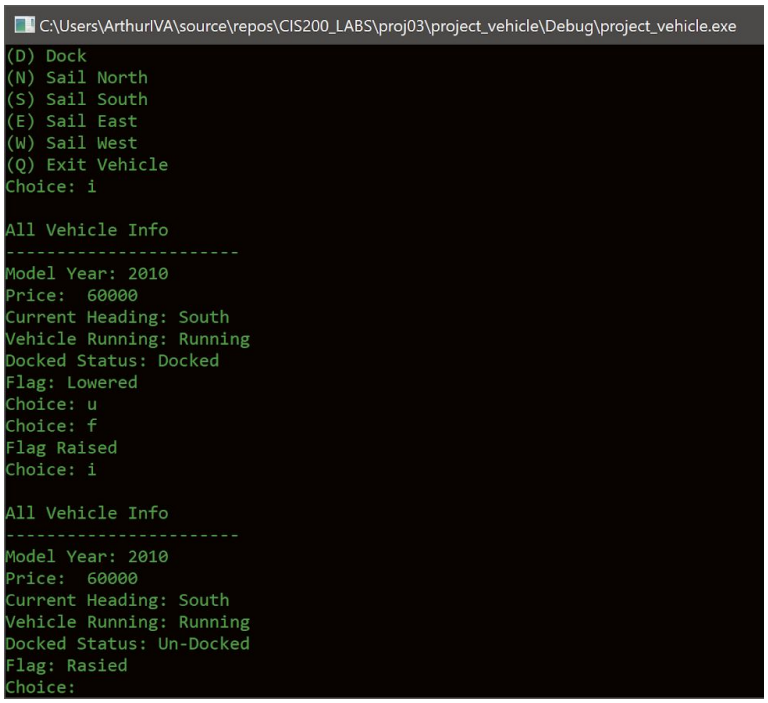
Valid Vehicle 11, 12, 13, 14.



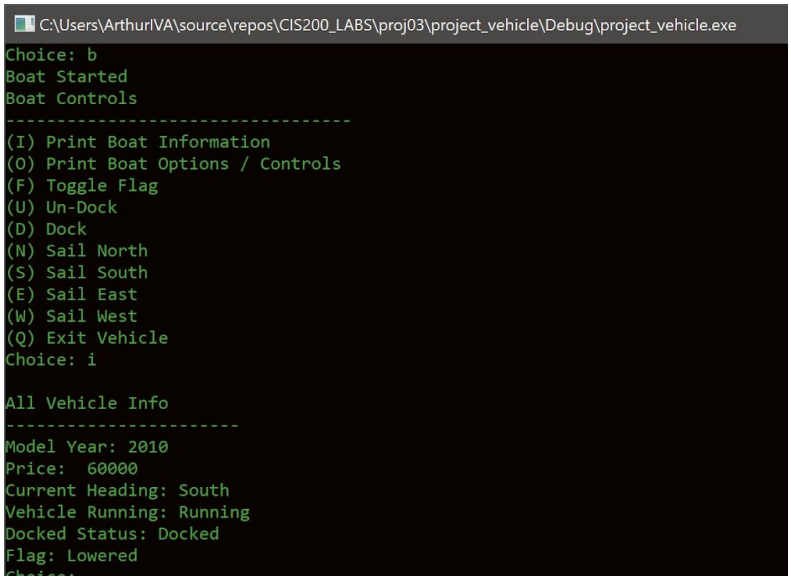
Invalid Vehicle 1, 2.



Valid Boat 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 13.

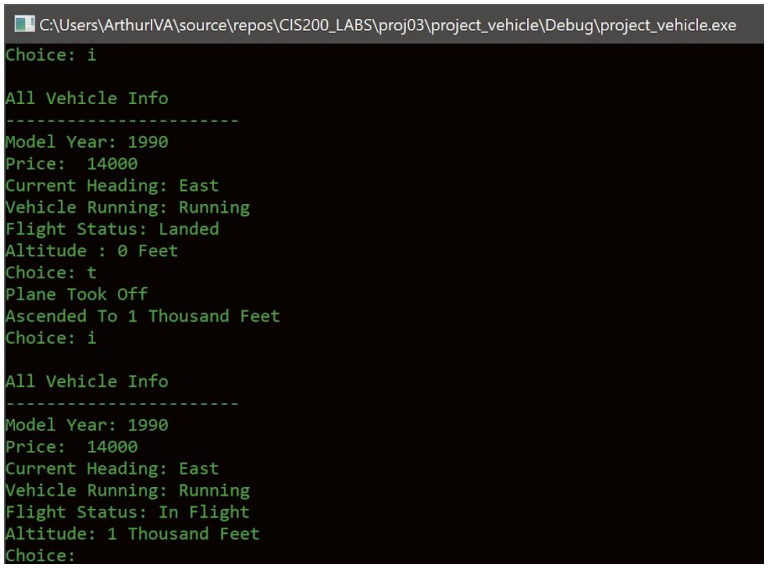


Valid Boat 1, 13.



Valid Boat 12, 14

Valid Plane 1, 2, 3, 4, 5, 6, 7, 8.



Valid Plane 9, 10, 12, 15, 16, 17, 19.

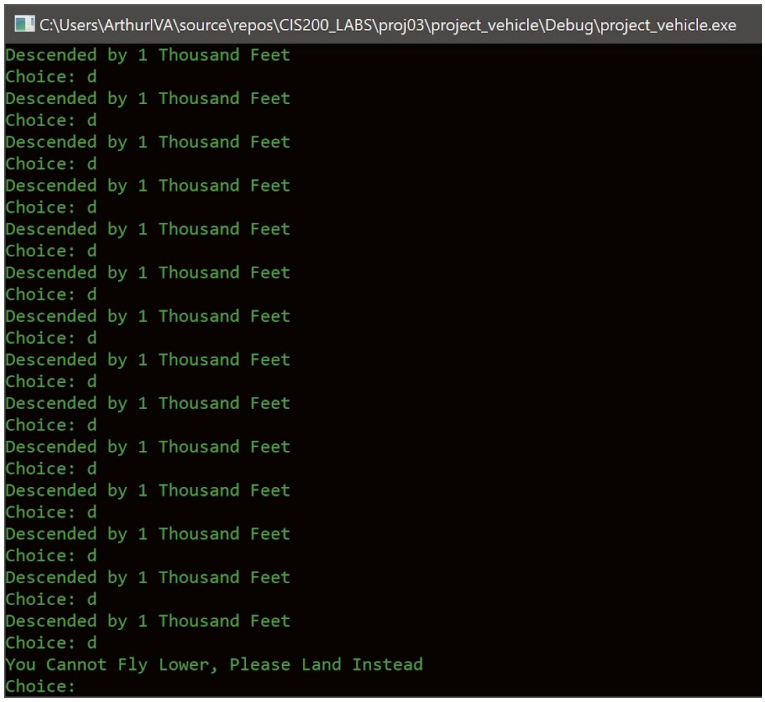
```
C:\Users\ArthurlVA\source\repos\CIS200_LABS\proj03\project_vehicle\Debug\project_vehicle.exe
(Q) Exit Vehicle
Choice: t
Plane Took Off
Ascended To 1 Thousand Feet
Choice: t
Plane Already Taken Off
Choice: l
Plane Landing From Current Altitude
Choice: l
Already Landed
Choice: t
Plane Took Off
Ascended To 1 Thousand Feet
Choice: a
Ascended by 1 Thousand Feet
Choice: i

All Vehicle Info
-----------------------
Model Year: 1990
Price:  14000
Current Heading: East
Vehicle Running: Running
Flight Status: In Flight
Altitude: 2 Thousand Feet
Choice:
```

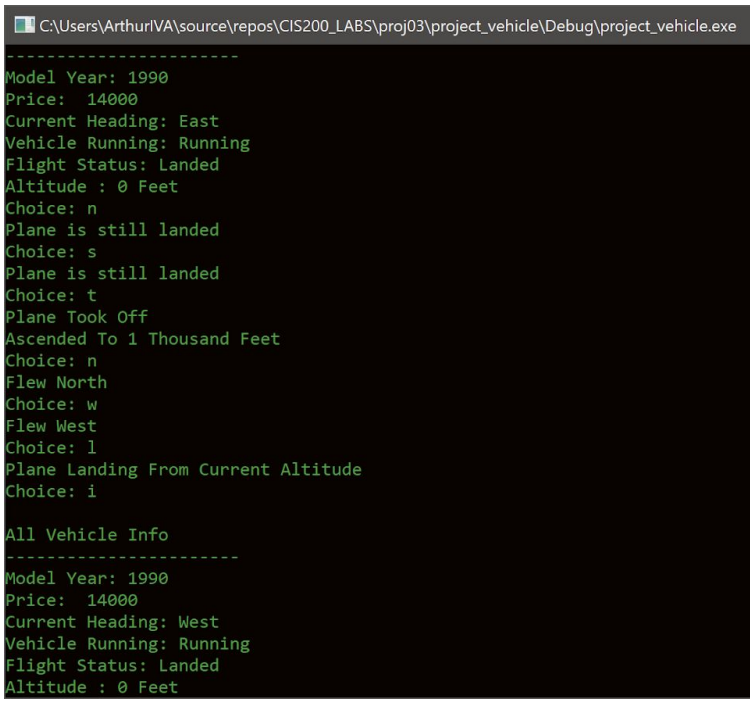
Valid Plane 11.

```
C:\Users\ArthurlVA\source\repos\CIS200_LABS\proj03\project_vehicle\Debug\project_vehicle.exe
Altitude: 2 Thousand Feet
Choice: a
Ascended by 1 Thousand Feet
Choice: a
Ascended by 1 Thousand Feet
Choice: a
Ascended by 1 Thousand Feet
Choice: a
Ascended by 1 Thousand Feet
Choice: a
Ascended by 1 Thousand Feet
Choice: a
Ascended by 1 Thousand Feet
Choice: a
Ascended by 1 Thousand Feet
Choice: a
Ascended by 1 Thousand Feet
Choice: a
Ascended by 1 Thousand Feet
Choice: a
Ascended by 1 Thousand Feet
Choice: a
Ascended by 1 Thousand Feet
Choice: a
Ascended by 1 Thousand Feet
Choice: a
Ascended by 1 Thousand Feet
Choice: a
Ascended by 1 Thousand Feet
Choice: a
Ascended by 1 Thousand Feet
Choice: a
Ascended by 1 Thousand Feet
Choice: a
You Cannot Fly That High
Choice:
```

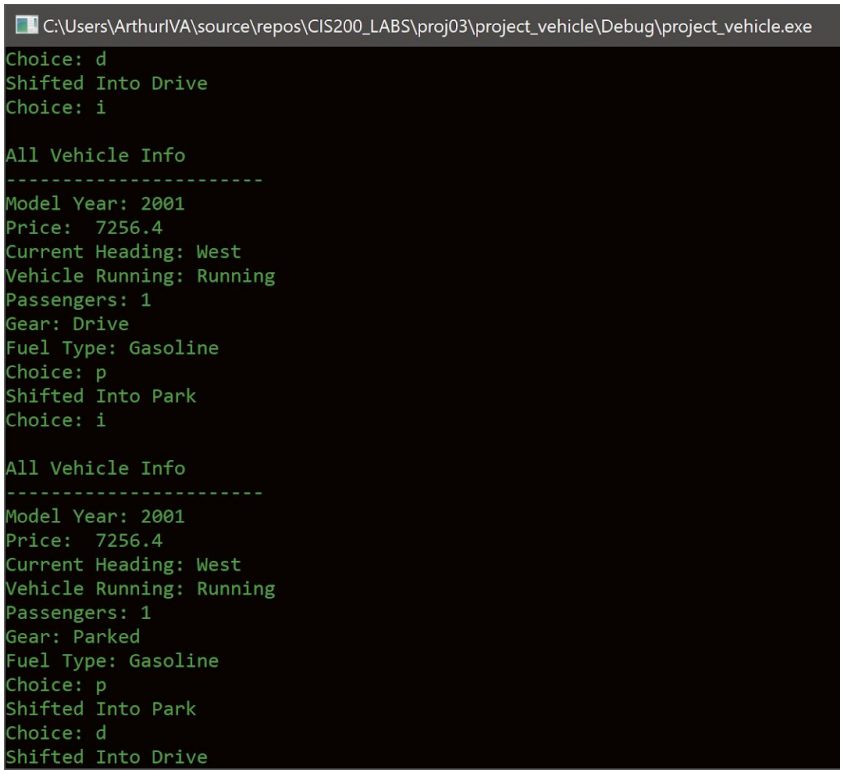
Valid Plane 14.



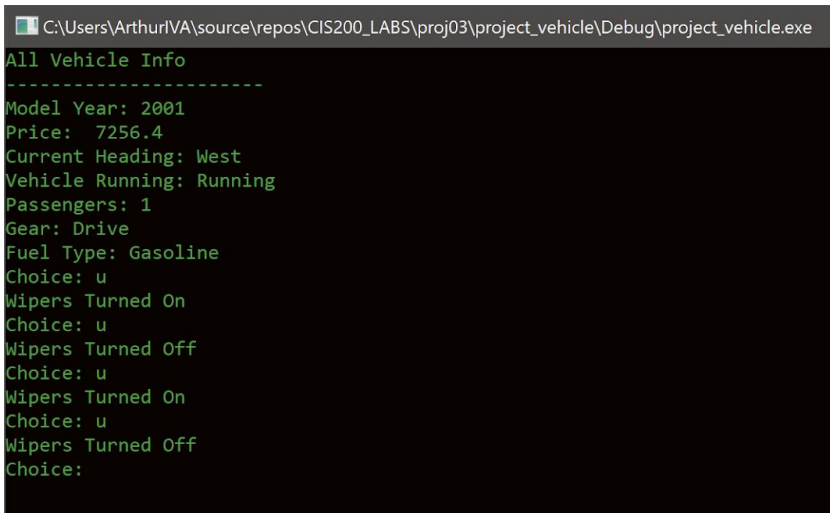
Valid Plane 20, 21.



Valid Land Vehicle 1, 2, 3.

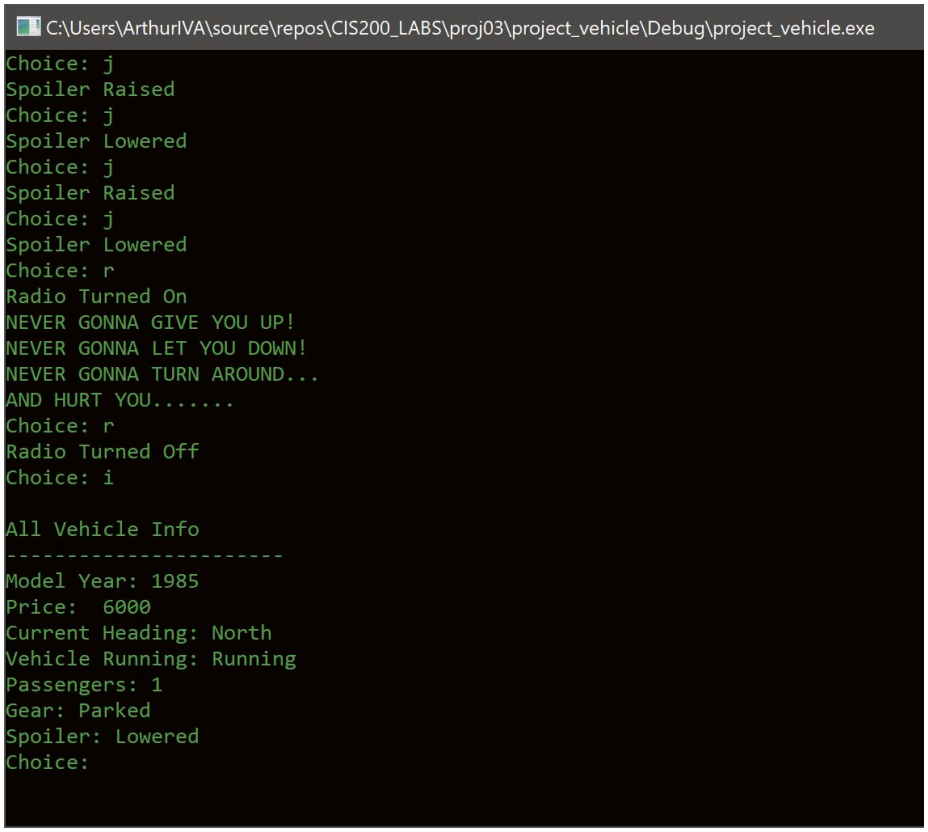


Valid Land Vehicle 4, 5, 6, 7, 8, 9, 10.



Valid Land Vehicle 12, 13.

Valid Car 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13.

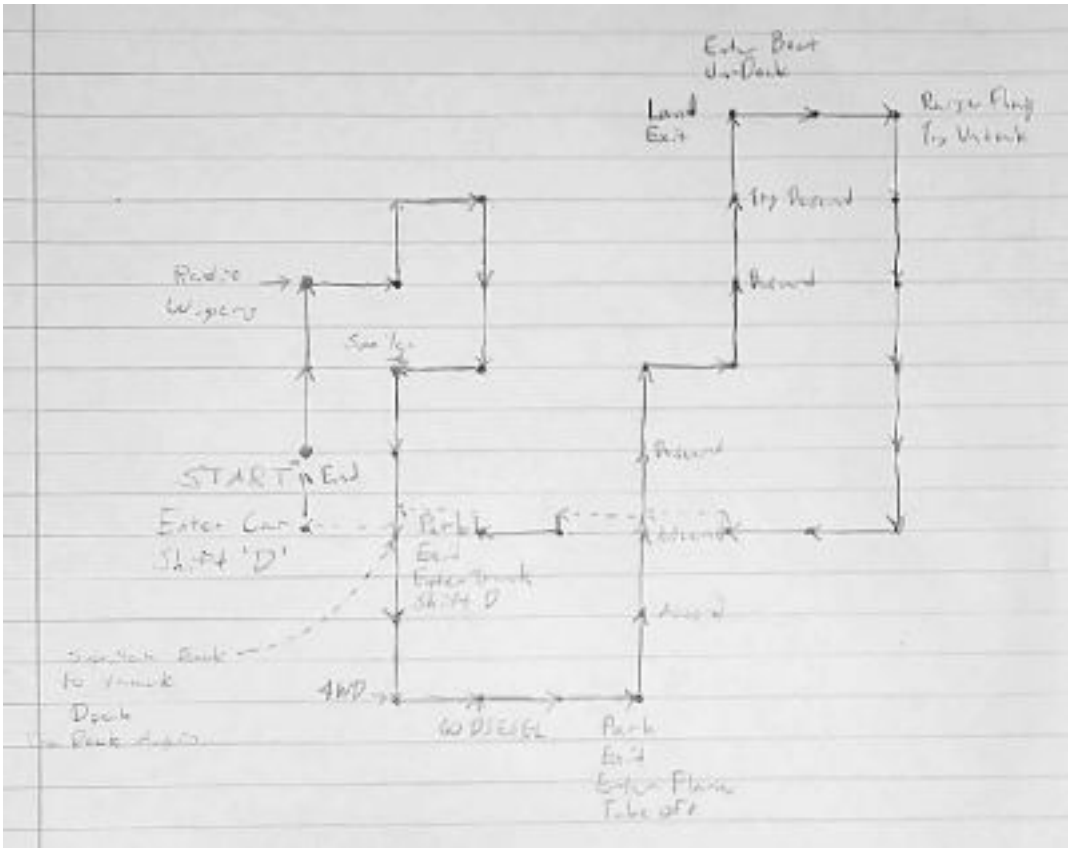


Valid Truck 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13.

```
C:\Users\ArthurIVA\source\repos\CIS200_LABS\proj03\project_vehicle\Debug\project_vehicle.exe
Choice: c
You're Running On Diesel Now. Go Roll Some Coal!
just kidding, don't do that, its a richard move
Choice: c
Silly Goose, You Can't Reverse That!
Choice: f
Four Wheel Drive Enabled
Choice: f
Four Wheel Drive Disabled
Choice: i

All Vehicle Info
----------------------
Model Year: 2001
Price:  7256.4
Current Heading: West
Vehicle Running: Running
Passengers: 1
Gear: Parked
Fuel Type: Yee Haw Diesel
4WD Status: Inactive
Choice:
```

Path



Log Part 1


log.txt - Notepad
File Edit Format View Help
Program Started
Started Trip
Car Started
Failed to Drive North
Shifted Into Drive
Drove North
Drove North
Toggled Radio
Toggled Wipers
Drove East
Drove North
Drove East
Drove South
Drove South
Drove West
Toggled Spoiler / Race Mode
Drove South
Drove South
Shifted Into Park
Turned Off Ignition
Exited Car
Ended Trip
Started Trip
Truck Started
Shifted Into Drive
Drove South
Drove South
Toggled 4WD
Drove East
Attempted to Change Diesel Status
Drove East
Drove East
Shifted Into Park
Turned Off Ignition
Exited Truck
Ended Trip
Started Trip
Plane Started
Took Off
Flew North
Ascended
Flew North
Ascended
Flew North
Descended
Flew North
Flew East
Flew North
Windows (CRLF)
Ln 88, Col 16
100%

Log Part 2

log.txt - Notepad

File Edit Format View Help

```
Ascended
Flew North
Descended
Flew North
Flew East
Flew North
Descended
Flew North
Failed to Descend
Flew North
Landed
Requested Plane Controls
Turned Off Ignition
Exited Plane
Ended Trip
Started Trip
Boat Started
Un-Docked
Sailed East
Sailed East
Toggled Flag
Failed to Un-Dock
Sailed South
Sailed South
Sailed South
Sailed South
Sailed South
Sailed West
Sailed West
Sailed West
Sailed West
Sailed West
Docked
Turned Off Ignition
Exited Boat
Ended Trip
Started Trip
Truck Started
Shifted Into Drive
Drove West
Drove North
Shifted Into Park
Turned Off Ignition
Exited Truck
Ended Trip
Exiting Program
```

Windows (CRLF) Ln 88, Col 16 100%

12. Error Log

Error Type (Logic/Runtime)	Cause of Error	Solution to Error
Logic	Using “==” During *toUpper*()	Remove second ‘=’.

13. Status

The smorgasbord works and it actually navigable. I’m proud to say that this went so much smoother than I anticipated because of planning the algorithm ahead of time.