The base for this assignment was very similar to the lab, so I was able to reuse a lot of the code I had written for the lab, and also reuse the exact same concepts as well. The differences started in having to do subclasses. I didn't realize until I was nearly done with the assignment that I was actually using polymorphism, and so that actually through me off a bit as I tried to make everything work. I kept having an error that the child function was not a member of the parent function, which it obviously isn't, but I didn't know I had to declare the child functions as virtual in the parent class. So in trying to fix that, I made my lists into arrays of pointers that pointed to other pointers in order to make an array of parent class with data that was of the subclasses. Once I fixed this, it all worked fairly well.

After that there was just some fine tuning to do, there were a few seg faults and the function to add another spot to the end of the array didn't copy stuff over right, I realized that when I was deleting the new list and moving stuff to the temporary one, I was deleting the data when I just needed to delete the pointer. I tried all the other functions and they worked, And I made input validation simple by just using utility functions for getting valid doubles and Ints. Because most the rest if my program runs on if/else statements, simply adding a “not valid option” the end if the user didn't enter a valid character for what to do worked as well.

I decided not to double the array when it was full for a couple of reasons. I already had most the code for just adding an extra spot every time. And also, if the array is massive, like it could be int this program, holding hundreds of cars, then doubling it could take up lots of storage. If the list was 40 GB then double the array would make the list 80 GB, and what if it's just to put a single other vehicle in? For the buying of cars, I didn't worry about taxes since Oregon doesn't charge taxes on vehicle purchases. I didn't know if Professor Rooker realized this, but it made it fairly easy. Other than that I just made it take in some stuff for subtracting from the price, and then deciding if you want to make payments.

For testing I tested adding vehicles to both arrays, entering unsatisfactory values for all, and then testing the different functions, such as searching for both a make or a price range, displaying both lists, buying and removing vehicles. If you enter the wrong character on most inputs, it will just kick you back to the prompt page, which I think is alright, as otherwise it would require a lot of while loops, and it doesn't take that long to get to wherever you were in the menu structure.

For a design plan, The last exercise was about designing the program, so I really just used that and followed it fairly accurately for designing the classes. I've attached the exercise.

Inventory

- create dynamic array (2 elements)
- add vehicle
  - expand array (double it)
- remove vehicle
  - shrink array

- search vehicle
  - getname()
  - ~~display vehicle info~~ display()
- display vehicle
  - get info from vehicles (in depth)
- display list
  - less info, full list
- list length

Main program

~~Automobile~~ Vehicle

lot #, make, model, color, mileage, fuel consumption, list price

Payments
- how many months (input)

Taxes
- list price * tax

Add Mileage
- change mileage

~~Print~~ getname()

Set stuff()

Print info()

Motorcycles

type (no street, moped, dirt)

setType()

~~Auto~~. Automobiles
setType
type(SUV, Pickup, convertable,
sedan)

Functions

display list

for(x < list length)
print information
(inventory[x].PrintInfo)

search
for(x< list length)
if(inventory[x].getName.find(input)=true
inventory[x].printInfo()

Payments
listprice/months

- Input validation:
  Making sure the ints are good & positive, same for doubles
  For strings, not really checking anything to check.

- test plan
  To test it, will first ~~need~~ to create a big list of vehicles & popwate it, checking input validation for the inputs. Next check search for things both in & not in the list. then removing a couple vehicles, looking for seg faults.
  No boundaries, in the program except maybe dates between 1900 & 2016 (variable). when entering a price over $10,000, it will prompt to confirm it