The objective of this lab is to:

Understand and practice polymorphism.

Instructions!

1. This is a **graded** lab, you are strictly **NOT** allowed to discuss your solutions with your fellow colleagues, even not allowed asking how is he/she is doing, it may result in negative marking. You can **ONLY** discuss with your TAs or with me.
2. You are strictly **NOT** allowed to discuss your solutions with your fellow colleagues, even not allowed asking how is he/she is doing, it may result in negative marking. You can **ONLY** discuss with your TAs or with me.
3. Follow good coding conventions.

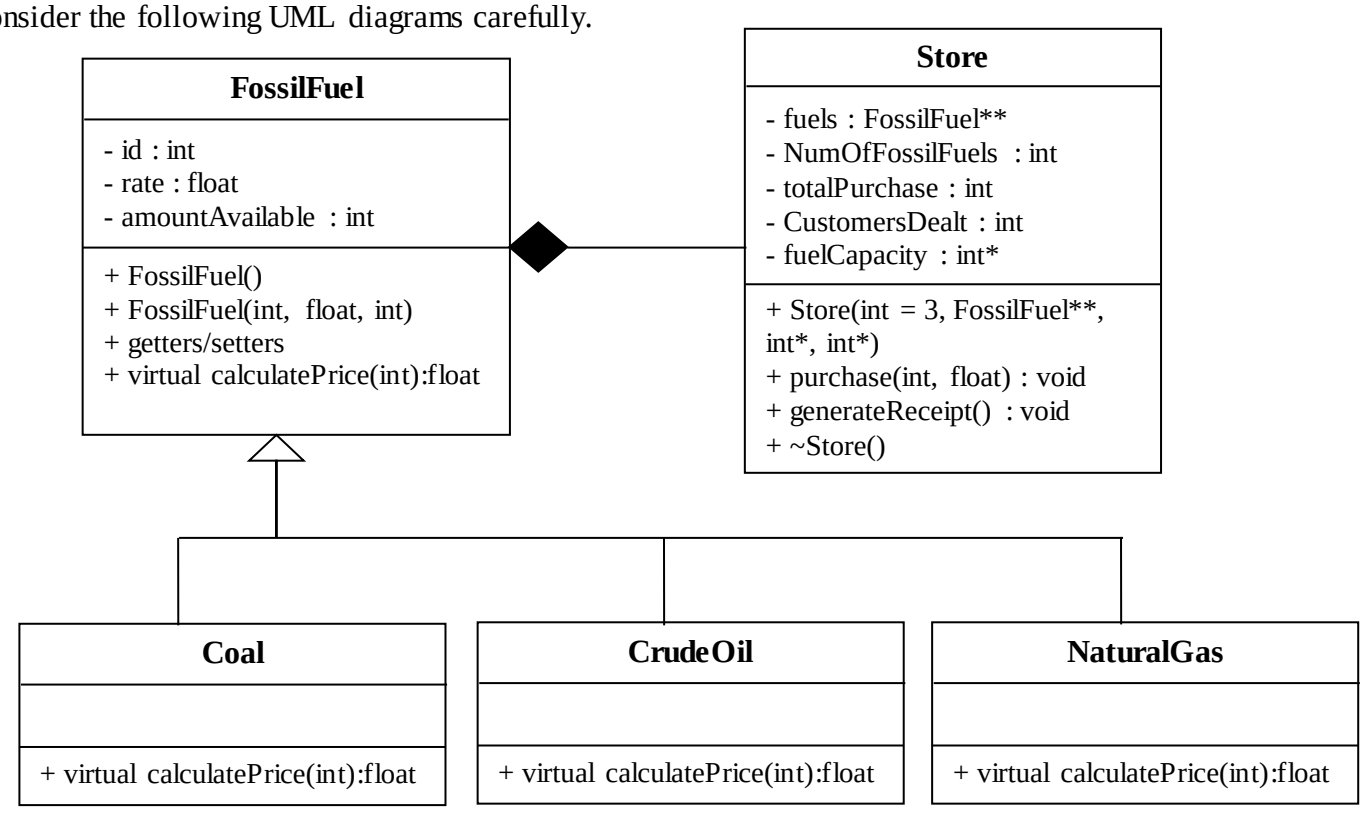
Task 01: [30 Marks]

The software you are going to host is an online low cost fuels retailer. Implementations require handling purchases and amounts with all the transactions incorporated into individual objects. Construction crew for your store installs a capacity to hold the amount of fuels in. There will be a specific amount of each fossil fuel that your store can hold. Regardful of that, there is also an amount of each fossil fuel that the store keeps. The customer comes to the store, specifies the amount as well as fossil fuel choice and after the price is paid, the amount of that particular fossil fuel is eventually deducted.

Your store is capable of giving services for three different fossil fuels. These are natural gas, crude oil, and coal. Their measuring units and rates are as follows:

- Natural Gas: $0.26 per therm
- Crude Oil: $55.0 per barrel
- Coal: $40.50 per short ton

Consider the following UML diagrams carefully.



The description of above entities is as follows:

FossilFuel

Data Members:

- **id:** It can either be 0, 1 or 2 as it represents the three fossil fuels. 0 for natural gas, 1 for crude oil and so on.
- **rate:** It is the rate, a floating point number which is different for each fossil fuel and will be set manually in the constructor of *Store* class.
- **amountAvailable:** An integer e.g. number of barrels, number of therms, etc.

Member Functions:

- FossilFuel (int id, float rate, int amountAvailable): The function will accept the parameters and apply same constraints on *id* as above. The *amountAvailable* should not be more than the *Store* can hold.
- You have to implement all the getters and setters for the three data members.
- virtual float calcPrice(float amount): The overridden functions of three base classes must calculate and return the price that user has to pay. The calculations will differ as the *rate* data member will differ in each sub-class.

Store

Data Members:

- **FossilFuel **fuels:** The array will hold the objects of fuels.
- **numOfFossilFuels:** The number of fossil fuels. Your store is capable of three so its value should be 3.
- **totalPurchase:** The total amount of money your sales has made you all over.
- **customersDealt:** The number of customers that came to purchase fuel from your store.
- **int *fuelCapacity:** An array, each of whose index will represent the *id* of a fuel. The value in each index will always be greater or equal to the corresponding fuel.

Member Functions:

- Store (int numOfFossilFuels = 3, FossilFuel **fuels, int *fuelCapacity, int *amountAvailable): The constructor should take the parameters and fit them in the appropriate data members. The size of array *fuelCapacity* as parameter will be same as the *numOfFossilFuels* in the input parameters. Same goes for *amountAvailable*. Take the values from each index of *amountAvailable* and assign to the data members of objects stored on each index of ***fuels*. The ***fuels* array itself will be populated in main() through polymorphism.
- purchase (int id, float amount): Of course, the user will have to input two parameters and the amount of the respective fuel will be deducted in this function. If the store runs out of the fuel that user demands, an apologizing message should be displayed.
- generateReceipt (): This function should display all the details of the purchase a customer proceeds with.
- ~Store(): Get rid of all the garbage!

Add a little feature to your price calculating functions. The customer should get a discount when he/she purchases equal to or more than a specified amount. The discount calculation should be done according to the following:

- 10% discount if purchase is worth $40.45 for natural gas.
- 15% discount if purchase is worth $98.99 for crude oil.
- 11% discount if purchase is worth $176.34% for coal.

Return the amount after applying the discount and also write a statement to show how much discount the customer got, the price before and the price after.