**A: DOCUMENTATION**

**Name and Title: Vehicle Survey Analysis**

**Introduction**

Templates in c++ is defined as a blueprint or formula for creating a generic class or a function. To simply put, you can create a single function or single class to work with different data types using templates. Templates in c++ work in such a way that it gets expanded at compile time, just like macros and allows a function or class to work on different data types without being rewritten.

**Requirements**

A news publication specializing in vehicle journalism conducts an annual survey of vehicles in the state to gather information about type of vehicles, their price and fuel options. The publication has a C++ based software with following classes/functionalities

**Vehicle template class**

* Data members

|  |  |  |
| --- | --- | --- |
| **Attribute** | **Data Type** | **Description** |
| **vehicleId** | Int or string | It indicates the vehicle number. The value can be an integer or a string value. |
| **vehicleBrand** | enum BRAND or string | It indicates the brand of the vehicle. It can be a value from the enum BRAND defined below or a string. |
| **vehicleRegistrationYear** | int | It indicates the year in which the vehicle was registered with the RTO department of the state. |
| **vehiclePrice** | int | It represents the price of the vehicle (in lacs). |
| **vehicleType** | Enum VEHICLE\_TYPE | It must be a value from the enum VEHICLE\_TYPE defined below and represents the type(category) of the vehicle. |

* Methods

|  |  |  |  |
| --- | --- | --- | --- |
| **Function Name** | **Arguments** | **Return**  **Type** | **Description** |
| **Constructor** | int or string,  enum BRAND or string,  int,  int  VEHICLE\_TYPE | none | Parameterized constructor to instantiate the class. |
| **getVehicleId** | none | String or int | It must return the *vehicleId* value stored in the current instance. |
| **getRegistrationYear** | none | int | It must return the *vehicleRegistrationYear* stored in the current instance. |
| **getVehicleBrand** | none | enum BRAND or string | It must return the *vehicleBrand* value saved in the current instance. |
| **getVehiclePrice** | none | int | It must return the *vehiclePrice* value stored in the current instance. |
| **getVehicleType** | none | enum VEHICLE\_TYPE | It must return the vehicleType value stored in the current instance |

**Functionalities**

* Methods

|  |  |  |  |
| --- | --- | --- | --- |
| **Function Name** | **Arguments** | **Return**  **Type** | **Description** |
| **findRegistrationYearById** | *vehicleId of type int or string,*  *standard vector of Vehicle objects* | int | The function must return the *vehicleRegistrationYear* of the vehicle object with the *vehicleId* matching the id passed as argument. If no object with a matching id is found, the function must return -1. |
| **countOfBrand** | *vehicleBrand of type string or enum BRAND,*  *standard vector of vehicle objects* | int | The function must return the number of instances found in the vector of objects with *vehicleBrand* matching the brand passed as first argument. |
| **averagePriceForGivenIds** | Standard vector of vehicleId values,  standard vector of vehicle objects | float | The function finds the objects with vehicleId values matching those passed as argument and returns the average price among such vehicles. |

**Enum**

**Enum VEHICLE\_TYPE**

* It must be either PETROL, DIESEL, EV or HYBRID.

**Enum VEHICLE\_BRAND**

* It must be either HONDA,TATA or MARUTI.

**Test cases:**

**\*\*PLEASE DO NOT EDIT THE TEST CASES**

The above-mentioned requirements have been considered for creating test cases.

The below table gives details of test cases to be used for validating written code.

|  |  |  |  |
| --- | --- | --- | --- |
| **Test**  **Case**  **No.** | **Test Case Description** | **Input Values** | **Expected Output** |
| 1. | To validate parameterized constructor functionality of *Vehicle* class. | Vehicle<int,VEHICLE\_BRAND> v1(1,TATA,2018,10,PETROL); | 1,TATA,2018,10  ,PETROL |
| 2 | To validate *findRegistratonYearById* functionality.. | Vehicle<int,VEHICLE\_BRAND> v1( 1,TATA,2018,10,PETROL);  Vehicle<int,VEHICLE\_BRAND> v2( 2,MARUTI,2016,12,PETROL);  Vehicle<int,VEHICLE\_BRAND> v3( 3,TATA,2018,15,EV);  Vehicle<int,VEHICLE\_BRAND> v4( 4,MARUTI,2018,8,PETROL);  Vehicle<int,VEHICLE\_BRAND> v5( 5,HONDA,2017,11,DIESEL);  std::vector<Vehicle<int,VEHICLE\_BRAND>> records;  records.push\_back(v1);  records.push\_back(v2);  records.push\_back(v3);  records.push\_back(v4);  records.push\_back(v5);  findRegistrationYearById<int,VEHICLE\_BRAND>(3,records); | 2018 |
| 3. | To validate *countOfBrand* functionality. | Vehicle<int,VEHICLE\_BRAND> v1( 1,TATA,2018,10,PETROL);  Vehicle<int,VEHICLE\_BRAND> v2( 2,MARUTI,2016,12,PETROL);  Vehicle<int,VEHICLE\_BRAND> v3( 3,TATA,2018,15,EV);  Vehicle<int,VEHICLE\_BRAND> v4( 4,MARUTI,2018,8,PETROL);  Vehicle<int,VEHICLE\_BRAND> v5( 5,HONDA,2017,11,DIESEL);  std::vector<Vehicle<int,VEHICLE\_BRAND>> records;  records.push\_back(v1);  records.push\_back(v2);  records.push\_back(v3);  records.push\_back(v4);  records.push\_back(v5);  countOfBrand<int,VEHICLE\_BRAND>(TATA,records); | 2 |
| 4 | To validate *averagePriceForGivenIds* functionality | Vehicle<int,VEHICLE\_BRAND> v1( 1,TATA,2018,10,PETROL);  Vehicle<int,VEHICLE\_BRAND> v2( 2,MARUTI,2016,12,PETROL);  Vehicle<int,VEHICLE\_BRAND> v3( 3,TATA,2018,15,EV);  Vehicle<int,VEHICLE\_BRAND> v4( 4,MARUTI,2018,8,PETROL);  Vehicle<int,VEHICLE\_BRAND> v5( 5,HONDA,2017,11,DIESEL);  std::vector<Vehicle<int,VEHICLE\_BRAND>> records;  records.push\_back(v1);  records.push\_back(v2);  records.push\_back(v3);  records.push\_back(v4);  records.push\_back(v5);    std::vector<int> ids;  ids.push\_back(1);  ids.push\_back(4);  ids.push\_back(5);  Float ans=averagePriceForGivenIds<int,VEHICLE\_BRAND>(ids,records); | 9.67  (Note: Value is rounded off to 2 decimal places for comparison/equality check) |