TIME ALLOWED: 3 HOURS MAX

Vehicle Rental Management System [100 Marks]

In this lab you have to implement a management system for your own startup "I love C++ Vehicle Rental Service!!!"

You need to implement seven classes, which are:

- Vehicle (Model, Make, Year, Reg#)
    - Cars (engineCapacity, SeatingCapacity, HourlyRent, MonthlyCost)
        - SportsCar (topSpeed, zero2hundredtime)
        - LargeCar (isOffRoad, CargoCapacity)
    - Trucks (CargoCapacity, DailyRent, YearlyCost)
        - LightTruck (hasRefrigerator, NameofLogoPrinted)
        - HeavyTruck (hasCrane, CrewCapcity)

Note: - From the structure above you should be able to understand which classes are derived from which other classes (the overall base class is the Vehicle class)

Task 1: Make the seven classes [20]

Make each of the classes mentioned above with the relevant fields and constructors.

Task 2: Import records for all vehicles from an input file [20]

This function requires that you import the records of all the university personnel from a text file (lab~~12~~_input.txt) with a given format. The format is as follows:

vehicle <make> <Model> <Year> <Reg#>
sportsCar <make><Model><Year><Reg#><engineCapacity><SeatingCapacity><HourlyRent><MonthlyCost><topSpeed><zero2hundredtime>
largeCar <make><Model><Year><Reg#><engineCapacity><SeatingCapacity><HourlyRent><MonthlyCost><isOffRoad><CargoCapacity>
trucks <make><Model><Year><Reg#><CargoCapacity><DailyRent><YearlyCost>
HeavyTruck <make><Model><Year><Reg#><CargoCapacity><DailyRent><YearlyCost><hasCrane><CrewCapcity>

Keep in mind that the file contains records of all different types of vehicles, and you have to read each record line by line, according to the type mentioned in the beginning of each line.

The catch is that you will use only one data structure (either a vector or an array of Vehicle pointers) to store all the vehicles. This vector will be storing elements of type *Vehicle. Keep in mind that a pointer

of a base class object can point to an object of its derived classes. This is essential for us, since we need to call virtual functions in the next few tasks. The file has been provided to you on LMS as well.

Task 3: View All Vehicles [20]

This requires that you develop a virtual function named void print() in the base class and the derived classes. So this task simply requires you to call the print function for all the elements in your vector. We should be able to see all the attributes of each of the vehicle on separate lines.

Task 4: View Total Yearly Profit of Rental Service [20]

This requires that you develop a virtual function named int income(). This will return the income (positive or negative) received from the specific vehicle instance. Try to keep in mind this is "yearly profit". You need to simply output the yearly profit of all individual vehicles and the entire rental service if this menu item is chosen. Note: A car is only rented out for an average of 2 hours per day.

Task 5: View Average Memory Required for Each Vehicle Record [20]

This requires that you develop a virtual function named int size(). Basically we need to know how much memory is used on average for each of our vehicle records. For this, define the size() function for each derived class (and base class). For this you can use the sizeof function in-built in C++. But you will have to understand how you can use it here, it is actually very simple. You have to decipher whether we can pass an instance of a user-defined class to the sizeof function, or do we just pass the class itself?

Note 1: Keep in mind that there may be a record of a vehicle which is neither a car nor an truck, so for such vehicles we still need to have our virtual function defined (for instance, since they add nothing to income, just return 0 for the income function). That is why it is important to implement our virtual functions for this lab for the base class as well, even if they don't make much sense.

Note 2: Obviously, lecture slides will be useful for reference.

Note 3: A description of sizeof function of C++ is given at the end of this lab manual.

The TAs have been asked to deduct marks if you ask very basic questions. Also if you ask TAs for solving basic syntax errors, you marks will be deducted. It is time that you learn to code independently and learn to understand the errors on your own.

New format of name in Lab submissions

<Student-ID>-Lab<#>-<TA-ID>.zip

For example 18100075-Lab4-Huzaifa.zip means this is the lab submission of 18100075 for Lab4 checked by Huzaifa. This way it will be easier for students and TAs to figure out the TA in case of any problems.

sizeof(*type* **)**	(1)
sizeof *expression*	(2)

Both versions return a constant of type std::size_t.

Explanation

1) returns size in bytes of the object representation of *type*.

2) returns size in bytes of the object representation of the type, that would be returned by *expression*, if evaluated.

Notes

Depending on the computer architecture, a byte may consist of 8 *or more* bits, the exact number being recorded inCHAR_BIT.

`sizeof(char)`, `sizeof(signed char)`, and `sizeof(unsigned char)` always return `1`.

sizeof cannot be used with function types, incomplete types, or bit-field glvalues.

When applied to a reference type, the result is the size of the referenced type.

When applied to a class type, the result is the size of an object of that class plus any additional padding required to place such object in an array.

When applied to an empty class type, always returns 1.

Keywords

`sizeof`

Example

The example output corresponds to a system with 64-bit pointers and 32-bit int.

```
#include <iostream>
struct Empty {};
struct Bit {unsigned bit:1; };
int main()
{
    Empty e;
    Bit b;
    std::cout << "size of empty class: "     << sizeof e        << '\n'
              << "size of pointer : "        << sizeof &e       << '\n'
//            << "size of function: "        << sizeof(void())  << '\n'  //
compile error
//            << "size of incomplete type: " << sizeof(int[])   << '\n'  //
compile error
//            << "size of bit field: "       << sizeof b.bit    << '\n'  //
compile error
              << "size of array of 10 int: " << sizeof(int[10]) << '\n';
}
```

Output:

```
size of empty class: 1
size of pointer : 8
size of array of 10 int: 40
```