## Вариант 1 (17.04.2021 г.)

В задачата можете да използвате наготово:

* класа std::string (заглавен файл <string>).
* Функциите от <cstring> – strlen, strcmp и т.н.
* Функциите от <cctype> – isalpha (проверява дали даден символ е буква от латинската азбука), isdigit (проверява дали даден символ е цифра).
* заглавен файл <stdexcept> (за работа с класовете за изключения).
* заглавен файл <cstddef> (за работа със std::size\_t).

В решението НЕ МОЖЕ да се използват контейнерите от STL (std::vector, std::list и т.н.).

За всеки от класовете НЕ МОЖЕ да се променя описаният интерфейс, но можете да добавите каквито прецените private функции или да разпишете допълнителни класове, които да ви помогнат в реализацията.

За всеки от класовете преценете дали трябва да реализирате функциите от rule of 3 (копиращ конструктор, копиращо присвояване, деструктор).

А) *(3 точки)* За улеснение, ще считаме, че всеки регистрационен номер на превозно средство е поредица от символи, подредени както следва:

1. Един или два символа от латинската азбука, които указват населено място;
2. Четири цифри;
3. Два символа от латинската азбука.

Например C1234AB, XY1111YX са валидни регистрационни номера, а 111145, ABC34DEF, ABCDEF и C11D не са.

Реализирайте клас Registration представящ регистрационен номер. Вътрешно класът да пази информацията като масив от тип char[9] съдържащ символите на номера. Класът да има следния интерфейс:

* Класът НЕ ТРЯБВА да има конструктор по подразбиране.
* Registration(const char\* str)  
  Конструктор, който зарежда в обекта регистрационния номер записан в str. Ако str не съдържа валиден регистрационен номер, хвърля изключение от тип std::exception.
* Registration& operator=(const char\* str)  
  Зарежда в обекта регистрационния номер записан в str. Ако str не съдържа валиден регистрационен номер, хвърля изключение от тип std::exception.
* bool operator==(const char\* str) const  
  Проверява дали два регистрационни номера са еднакви.
* const char\* toString() const  
  Връща регистрационния номер като C-style символен низ.

Б) *(1 точка)* Реализирайте клас Vehicle (превозно средство). То трябва да има следните публични член-променливи:

* regnum – регистрационен номер, константа от тип Registration.
* description – описание, константа от тип std::string.

Класът да има следния интерфейс:

* Класът ДА НЯМА конструктор по подразбиране
* Vehicle(const char\* regnum, const char\* description)

В) *(5 точки)* Реализирайте клас VehicleList представящ списък от коли.

Списъкът има капацитет, който се указва при неговото създаване. Капацитетът е от тип std::size\_t и може да бъде произволно голям (помислете какво значи това за решението; как трябва да осигурите паметта). Той се указва само веднъж при създаването на списъка и после не може да се променя.

В списъка не трябва да може да се съдържат две превозни средства с еднакви номера.

Обърнете внимание, че Vehicle обектите нямат default constructor. Преценете как да адресирате този проблем (например използвайте масив от указатели).

Класът да има следния интерфейс:

* VehicleList(std::size\_t capacity)  
  създава списък, който може да съдържа най-много capacity на брой превозни средства.
* Всички функции от rule of 3 (по желание: всички от rule of 5). Забележете, че VehicleList не е като класа за гараж от домашното. VehicleList притежава колите съхранени в него и трябва да ги почиства в деструктора си. При копиране на списък, новото копие трябва да създаде за себе си нови обекти от тип Vehicle; то не трябва да сочи към тези на оригинала. Копието трябва да бъде със същия капацитет като оригинала. Операторът за присвояване да дава strong exception safety guarantee.
* void insert(const char\* regnum, const char\* description)  
  добавя превозното средство с регистрационен номер regnum и описание description в списъка. Ако операцията не успее (например няма повече място, регистрационният номер е невалиден, такъв регистрационен номер вече се съдържа в списъка), да се хвърля изключение от тип std::exception. Операцията да дава strong exception guarantee.
* const Vehicle& at(std::size\_t index) const  
  достъп до елемента намиращ се на позиция index. Ако такъв няма, да се хвърля изключение std::out\_of\_range.
* const Vehicle& operator[](std::size\_t index) const  
  достъп до елемента намиращ се на позиция index. Функцията да не прави проверка за коректност дали index е валидна позиция. (В debug режим assert-вайте дали index е валидна позиция).
* bool empty() const  
  Проверява дали списъка е празен (т.е. в него не е било добавено нито едно добавено превозно средство).
* std::size\_t capacity() const  
  капацитет на списъка.
* std::size\_t size() const  
  брой превозни средства добавени в списъка.
* const Vehicle\* find(const char\* regnum) const  
  намира и връща превозното средство с регистрационен номер regnum. Ако такова няма, да се върне nullptr.

Г) *(3 точки)* Напишете програма, която:

1. Въвежда от потребителя число N и след това създава списък от превозни средства с дължина N.
2. Въвежда от потребителя точно N коли и ги запазва в списъка. Ако потребителят въведе некоректен регистрационен номер или опита да въведе един и същ номер два пъти, програмата ви трябва да може да улови хвърленото изключение. В такъв случай се извежда съобщение за грешка и потребителят може отново да опита да въведе данните за колата.
3. Позволява на потребителя да въведе регистрационен номер на кола и отговаря дали тя се съдържа в списъка или не.