Софийски университет „Св. Климент Охридски“
Факултет по математика и информатика

Домашно 1

курсове *Обектно-ориентирано програмиране и*
Обектно-ориентирано програмиране-практикум

специалности *Информатика и Компютърни науки (1-ви поток)*

летен семестър 2020/21 г.

Редакции

2021-04-01 – добавено е пояснение за класа `Vehicle`, че членовете трябва да са от тип `MyString`. Добавено е пояснение, че не могат да се използва `std::string`, нито контейнерите от STL (`std::vector`, `std::list` и т.н.)

2021-04-03 – `Garage::insert` трябва да приеме `Vechile& v`, а не `const Vehicle& v`

2021-04-05 – Добавени са редица уточнения по условието, по въпроси от чата и от форума. За уточненията е създаден отделен раздел.

- уточнение за интерфейсите на `Vehicle` и `Garage`.
- уточнение за това, че не трябва да се държи отчет на паркоместата в гаража, а само на заетия обем.
- уточнение за това, че животът на обектите от тип `Vehicle` не зависи от този на гаража, в който те се добавят.
- в описанието на `Garage::clear` е дадено по-подробно уточнение какво прави функцията.
- в описанието на `Garage::erase` е уточнено, че при премахване на кола от гаража е допустимо да се промени редът на колите в него.
- За `Garage::size` е уточнено, че се има предвид да връща броя коли в него.
- За изискването за rule of 3 в символния низ и гаража е добавено допълнително, по-дълго описание. Възможно е да реализирате тези класове с помощта на други и така да не се наложи да следвате rule of 3 или rule of 5, а rule of 0. Това вече е изрично посочено в условието.

Коригиран е и един проблем в условието: От интерфейса на гаража са премахнати функциите, които директно връщаха обектите превозни средства, които се пазят в

него (тези функции не трябва да се реализират). Оставени са само константните им еквиваленти. Чрез тези функции беше възможно (чрез присвояване) да се подмени един `Vehicle` обект с друг, който има различен капацитет и по този начин да се заобиколят правилата на гаража.

Допълнителни уточнения

1. За `Vehicle` и `Garage` пише точно какъв да е интерфейсът, там не бива да се добавят други функции освен посочените. Оттук следва, че колата е immutable object, т.е. не може да си променя състоянието след като бъде създадена. За нея може да напишете unit test, който проверява дали ако създадете кола с параметри a,b,c (все едно сте я създали като `Vehicle(a,b,c)`), то функциите от интерфейса на колата връщат точно a,b,c, а не някакви други данни.

2. Гаражът няма нужда да пази кои точно паркоместа заема дадено превозно средство. Той се интересува само от това какъв обем е зает. Тоест ако добавите кола заемаща три места, няма нужда да се търсят три непрекъснати клетки в някакъв масив, а само да се провери дали е свободен съответният обем.

3. Превозните средства съществуват независимо от гаража. Той ги регистрира в себе си, но нито ги създава, нито има грижата да ги унищожава. Същевременно ако даден `Vehicle` обект прекрати съществуването си преди да е бил унищожен гаража, това може да предизвиква проблем. Затова трябва да се гарантира, че превозните средства няма да се унищожат докато гаражът все още съществува.

4. Като следствие от по-горното – гаражът НЕ БИВА да копира обектите в себе си, а да пази някакъв вид препратка (например указател) към превозните средства, които се пазят в него.

Условие на задачата

ВАЖНО: Покрийте всички класове с подходящи unit test-ове. Всички член-функции трябва да са добре тествани.

В решението на задачата НЕ МОЖЕ да се използва `std::string`, нито контейнерите от STL (`std::vector`, `std::list` и т.н.)

А) Реализирайте клас `MyString`, който представя символен низ. Класът да има поне следните операции:

- Default constructor.
- Гарантирайте, че обектите могат да се копират и унищожават коректно. Ако е нужно, реализирайте експлицитно всички функции от rule of 3 (бонус: гарантирайте, че работят коректно всички от rule of 5). Ако това не е нужно, следвайте rule of 0.
- `MyString(const char* str)` -- създава нов обект и копира в него съдържанието на `str`.
- `char& at(std::size_t pos)` -- достъп до елемента намиращ се на позиция `pos`. Ако такъв няма, да се хвърля изключение `std::out_of_range`.
- `const char& at(std::size_t pos) const` -- Като по-горното, но за константи.
- `char& operator[](std::size_t pos)` -- достъп до елемента намиращ се на позиция `pos`. Функцията да не прави проверка за коректност дали `pos` е валидна позиция. (В debug режим `assert`-вайте дали `pos` е валидна позиция).
- `const char& operator[](std::size_t pos) const` -- Като по-горното, но за константи.
- `char& front()` -- достъп до първия символ в низа. Да НЕ СЕ прави проверка за коректност дали такъв символ има. (В debug режим `assert`-вайте че низът не е празен).
- `const char& front() const` -- Като по-горното, но за константи.
- `char& back()` -- достъп до последния символ в низа. Да не се прави проверка за коректност дали такъв символ има. (В debug режим `assert`-вайте че низът не е празен).
- `const char& back() const` -- Като по-горното, но за константи.
- `bool empty() const` -- Проверява дали низът е празен.
- `std::size_t size() const` -- дължина на низа.
- `void clear()` -- изчиства съдържанието на низа.
- `void push_back(char c)` -- добавя символа `c` в края на низа. Операцията да дава strong exception guarantee.
- `void pop_back()` -- премахва последния символ от низа. Да не се прави проверка за коректност дали такъв символ има. (В debug режим assert-вайте че низът не е празен).
- `MyString& operator+=(char c)` -- добавя символа `c` в края на низа. Операцията да дава strong exception guarantee. Връща `*this`.
- `MyString& operator+=(const MyString& rhs)` -- конкатенира съдържанието на `str` към текущия низ. Операцията да дава strong exception guarantee. Връща `*this`.
- `MyString operator+(char c) const` -- Връща нов символен низ, който се получава от текущия, конкатениран със символа `c`.

- `MyString operator+(const MyString& rhs) const` -- Връща нов символен низ, който се получава от текущия, конкатениран с низа `rhs`.
- `const char* c_str() const` -- връща указател към null-terminated масив от тип `char`, който има съдържание идентично с това на низа.
- `bool operator==(const MyString &rhs) const` -- Проверява дали два символни низа са еднакви.
- `bool operator<(const MyString &rhs) const` -- Проверява дали текущият низ предхожда лексикографски `rhs`.

Б) Реализирайте клас `Vehicle` (превозно средство). То трябва да има следните свойства:

- регистрационен номер -- символен низ (използвайте типа `MyString`).
- описание -- символен низ (използвайте типа `MyString`).
- брой места за паркиране, които заема превозното средство -- стойност от тип `std::size_t`.

Класът да има следния интерфейс:

- Класът ДА НЯМА default constructor
- `Vehicle(const char* registration, const char* description, std::size_t space)`
- `const char* registration() const` -- Връща регистрационния номер като C-style символен низ.
- `const char* description() const` -- Връща описанието на превозното средство като C-style символен низ.
- `std::size_t space() const` -- Връща мястото, което заема превозното средство при паркиране.

В) Реализирайте клас `Garage` (гараж). Гаражът има капацитет, който се указва при неговото създаване. Капацитетът е стойност от тип `std::size_t`, която може да бъде произволно голяма (помислете какво значи това за решението; как трябва да осигурите паметта).

В гаража трябва да може да се добавят и премахват превозни средства. Трябва обаче да се отчита неговият капацитет и броят на вече добавните превозни средства. Например в гараж с капацитет 3 трябва да може да се сложат най-много:

- три превозни средства заемащи едно място
- едно превозно средство заемащо две места и едно заемащо едно място
- и т.н.

В такъв гараж не може едновременно да се съдържат две превозни средства, всяко от които заема две места; едно превозно средство заемащо пет места и т.н.

Освен това, в един гараж не може да има две коли с еднакви регистрационни номера.

Тези изисквания трябва да се отчитат при добавянето на нови превозни средства в гаража.

Класът да има следния интерфейс:

- `Garage(std::size_t size)` -- създава гараж с максимално място за паркиране `size`.
- Гарантирайте, че обектите могат да се копират и унищожават коректно. Ако е нужно, реализирайте експлицитно всички функции от rule of 3 (бонус: гарантирайте, че работят коректно всички от rule of 5). Ако това не е нужно, следвайте rule of 0.
- `void insert(Vehicle& v)` -- добавя превозното средство `v` в гаража. Ако операцията не успее (например няма достатъчно памет, в гаража няма повече място за паркиране, вече има кола със същия регистрационен номер), да се хвърля изключение. Операцията да дава strong exception guarantee.
- `void erase(const char* registration)` -- премахва колата с регистрационен номер registration от гаража. Ако такава няма, да не се прави нищо. При премахването на кола от гаража е допустимо да се промени редът на останалите в гаража. Това условие ще ви позволи при премахване на елемент да поставите последния елемент от масива на мястото на премахнатия, вместо да правите left shift.
- `const Vehicle& at(std::size_t pos) const` -- достъп до елемента намиращ се на позиция `pos`. Ако такъв няма, да се хвърля изключение `std::out_of_range`.
- `const Vehicle& operator[](std::size_t pos) const` -- достъп до елемента намиращ се на позиция `pos`. Функцията да не прави проверка за коректност дали `pos` е валидна позиция. (В debug режим `assert`-вайте дали `pos` е валидна позиция).
- `bool empty() const` -- Проверява дали гаражът е празен.
- `std::size_t size() const` -- брой елементи (превозни средства) в гаража.
- `void clear()` -- изчиства съдържанието на гаража. Това означава, че в него не се съдържа нито една кола. Капацитетът му обаче остава непроменен. Така в него могат отново да се добавят нови коли.

- `const Vehicle* find(const char* registration) const` -- намира и връща превозното средство с регистрационен номер `registration` в гаража. Ако такова няма, да се върне `nullptr`.

Г) Напишете програма, която позволява на потребителя да създаде гараж с избран от него капацитет. След това програмата трябва да позволи в него да се добавят и премахват превозни средства. Да има и операция, която извежда съдържанието на гаража на екрана.