`

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

ЗАДАЧИ ЗА САМОСТОЯТЕЛНА РАБОТА

курс Обектно-ориентирано програмиране
за специалности Информатика, Компютърни науки и Софтуерно инженерство

Задача 1. Напишете програма, която ще представлява интерфейс към файлова база от данни за играчи на Magic The Gathering.
Имате 3 **бинарни** файла:

- Файл с Играчи (players.dat):
  - Всеки играч си има име и уникално ID. Името на играча е не повече от 127 символа.
- Файл с Карти (cards.dat):
  - Всяка карта има име, ID и цвят. Името на картата е не повече от 63 символа, а цвета е един от { Red, Black, Blue, White, Green }.
- Файл с Тестета (decks.dat):
  - Всяко тесте има ID на притежател (играч), цвят и ID-та на картите. Цвета е един от { Red, Black, Blue, White, Green }; има точно 60 на брой ID-та на карта.

Във всеки един от файловете ID е цяло неотрицателно число, по малко от 2^15. За цвета на картите и тестетата изберете подходящ тип!

Програмата дава възможност за изпълнение на следните команди (ако се въведе невалидна команда програмата приключва):

- Създаване на ново тесте (команда `cd`) като са нужни име на притежател и 60 имена на картите в тестето, цвета на тестето автоматично е цвета, който най много се среща сред избраните карти. Операцията е успешна ако има играч с подаденото име и всички подадени имена на карти са валидни (съществува

запис за тях). Пример

```
cd
The Almighty Dragon
Stream of Life
Tranquility
Tsunami
Forcefield
Swamp
...
Lord of the Pit
```

- Създава нов играч (команда `cp`) по подадено име, ID-то на играча се генерира така че да няма друг играч с такова ID. Пример:

```
cp
The Almighty Dragon
```

- Създава нова карта (команда `cc`) по подаден цвят и име, ID-то се генерира така че да няма друга карта със същото. Пример:

```
cc
White Celestial Crusader
```

- Генерира текстов файл (команда `report`) с данни за всчиките тестета групирани по цвят и накрая извежда най-често използваната карта както и нейният цвят. Данните за тестетата са името на притежателя, името на първата карта и цвета и. Всяко тесте се принтира на нов ред. Пример:

```
report
```

При приключване на програмата бинарните файлове остават в адекватно (не счупено) състояние. Създадените играчи, карти и тестета се записват в съответните файлове преди програмата да приключи. При стартиране на програмата, се проверява за наличност на 3те бинарни файла и командите за създаване на играчи и карти взимат под внимание вече наличните такива!

Примерен `report` файл:

```
Red
Black
    Owner: Swamp Reacher Card: Syndicate Enforcer Color: Black
    Owner: Swamp Reacher Card: Dying Wish Color: Back
    Owner: Swamp Reacher Card: Basilica Guards Color: White
Blue
White
Green
    Owner: _*PeaceMaker*_ Card: Borderland Range Color: Green
    Owner: PiCkAcHu!11!one!one!1eleven! Card: Phyrexian Rebirth Color: White
    Owner: Swamp Reacher Card: Forest Color: Green
```

```
Most common card: Island Color: Blue
```

Задача 2. Както всички знаем идва Великден, а покрай Великден зайците имат много работа... Заекът Ставри е 3-кратен сребърен медалист по събиране на яйца и тази година е решен да спечели златото. От вас той иска да му направите кошница за състезанието (само с най-добрата кошница може горкият Ставри да се надява да триумфира).
По регламент всяко яйце се описва от уникален низ с точна дължина и неговата големина. Тъй като е състезание и всичко си има правила, той не може да използва каква да е кошница... Всяка кошница си има име и то е името на нейният собственик. Кошницата, която ще му направите трябва да може да променя размера си според това колко яйца има в нея - а това както сами предполагате е доста трудно. За това във всеки един момент, ако в нея има N яйца, то тя може да има свободно място за най-много още N яйца. Кошницата, която ще му осигурите трябва да позволява добавянето на ново яйце и премахването на яйце по даден низ, който го описва.

Всяка вечер кошниците на участниците трябва да се виртуализират някъде, затова от вашата програма, Ставри иска да може да сериализира кошницата си в бинарен формат (масив), като заеманото място е минимално, тоест низовете заемат точно толкова, колкото са дълги или константно число байтове повече. А на сутринта прогамата ви ще трябва да десериализира кошницата от бинарния формат.

В края на състезанието съдиите трябва да могат да преценят кой печели, затова трябва да предоставите функция която по зададена кошница генерира репорт за нейното съдържание. Ще трябва да го запишете в текстов файл с име : "report_nameOfBasket.txt", където nameOfBasket е името на кошницата за която генерирате репорт в момента. Вътре трябва да са описани всички яйца (чрез уникалния низ и тяхната големина). Всяко яйце трябва да се намира на нов ред.

Задача 3. С вашите кошници по вашата глава...

Добре познатият ви Ставри спечели това пусто злато. Той искрено ви благодари. Но... сега реши да отвори бизнес, защото чул, че от математиката се правели добри пари. Ще продава математически кошници, представете си... И разбира се пак вие трябва да спасявате положението - Той ви моли да добавите следните оператори към кошницата, която му направихте:

- Оператор [] - който по зададен индекс, ако той е валиден, връща яйцето, което се намира там.
- Оператор [] - който по зададен символен низ, връща първото яйце, което го съдържа, ако има такова.
- Оператори +, += - които конкатенират 2 кошници (работят върху аргументи две кошници).
- Оператори +, += - които конкатенират низ с всички яйца в дадена кошницата. (два аргумента - кошница и низ)
- Оператори *, *= - приемат цяло число и умножават всички размери на яйца в кошницата по това число. (аргументи кошница и число)
- Оператори /, /= - приемат цяло число и делят всички размери на яйца в кошницата на даденото число. (аргументи кошница и число)
- Оператори %, %= - които пресмятат сечението на кошници (работят върху аргументи две кошници).
- Оператори (==, !=, <, <=, >=) - сравнява две кошници, като две кошници са равни, когато всеки елемент на позиция i в първата кошница е равен на елемента на същата позиция във втората. А първата кошница е по-малка от втората, когато слещнете елемент в първата, който е по-малък от съответния във втората и до момента не сте срещнали по-голям елемент в първата. Две яйца са равни едно на друго ако низовете в тях са еднакви, а едно яйде е по-малко от друго яйце, ако низа на първото яйце е по-малък в лексикографски смисъл от низа във второто.
- За яйцата трябва да дефинирате поне 10 оператора, по ваш избор. Подберете ги така, че да улеснят максимално реализацията на кошница.
- Напишете и програма (main и други функции) с която да покажете използването на операторите.

Понеже Ставри вече е спечелил кошницата, която му написахте сега се изисква да я надградите (без да губите функционалност). Т.е. имплементацията на домашно 3 не трябва да премахва нищо, което е работело в домашно 2. Можете да преправята или дори да пренаписвате вашето домашно 2, ако надграждането не е възможно. И не забравяйте да съблюдавате добрите практики при писането на кода - все пак ще се продава ;)

Задача 4. Напишете програма, която ще представлява база от данни за притежатели на коли, съхранявайки следните данни:

- Име на притежателя (не по дълго от 23 символа)
- Уникален идентификационен номер на колата
- Регистрационен номер на колата (неотрицателно четирицифрено число)

Колите, които могат да бъдат притежавани от хората, са:

име	идентификационен номер	мощност
Lambordgini Murcielago	0	670
Mercedes-AMG	1	503
Pagani Zonda R	2	740
Bugatti Veyron	3	1020

Вашата програма трябва да поддържа следните функции:

- Добавяне на нов запис за притежател на кола с име, регистрационен номер на колата и идентификационен номер на колата. Ако в програмата съществува запис за кола със същия регистрационен номер, извеждате съобщение за грешка и не добавяте записа.
- Генериране на текстов файл с име "car-report.txt" и следната информация:
  - Най-популярна кола - най-много хора я притежават.
  - Средна мощност на всички притежавани коли.
  - По един ред за всеки притежател с името и общата мощност на притежаваните от него коли.
- Генериране на бинарен файл с име "db-save.dat", в който седят записи за всички притежатели на коли. Ако файлът вече съществува, се презаписва.
- При пускане, да зарежда притежатели на коли от бинарен файл с име "db-save.dat", ако такъв не съществува все едно е бил празен.

Забележки:

- Един човек, може да притежава повече от една кола - за програмата един и същи човек е такъв, който присъства няколко пъти с еднакво име.
- Не качвайте бинарни файлове генерирани от вашата среда за разработка.

Задача 5. Напишете програма която да представлява буркан за бисквитки. Вашият буркан трябва да може да побира произволен брой бисквитки, и свободното място в него да не е повече от бисквитките в него. От буркана могат да се ядат бисквитките една по една, както и да се добавят една по една. Всяка отделна бисквитка в буркана ще представлява множество от низове.
Вашата програма ще трябва да може да сериализира буркан с бисквитки в бинарен формат (масив) като заеманото място е минимално, тоест низовете заемат точно толкова колкото са дълги или константно число байтове повече. Също така ще трябва да може да десериализирате буркан с бисквитки от бинарен формат.

Напишете примерна main функция която демонстрира използването на буркана с бисквитките, както и сериализацията и десериализацията на бурканите.

Забележки

- Броят на низовете във всяка бисквитка може да е константо зададен при компилация.
- Низовете в бисквитките трябва да се съхраняват, в динамично заделена памет, от обекта (бисквитката) в който се намират и той да се грижи за нейното управление. Заделената памет за всеки низ трябва да е колкото е неговата дължина + 1 за терминиращата нула.

Задача 6.1. Имплементирайте клас който представлява N-мерен вектор, като N може да бъде **произволно** голямо (**int**). Всеки от компонентите на вектора е от тип дробно число (**double**). Имплементирайте следните оператори:

- Оператор [] който приема аргумент цяло число индекс, и връща съответният компонент на този индекс, при достъпване на грешен компонен се връща 0.
- Оператори +,-,+=,-= извършват събиране и изваждане на вектори с еднаква размерност.
- Оператор *,*= извършват умножение на вектор с число (скалиране).
- Оператор /,/= извършват деление на вектор с число (скалиране).
- Оператор % извършва скаларно произведение на вектори с еднаква размерност.

При извикване на оператор, който изисква вектори с еднаква размерност, ако векторите са с различни размери, по-малкият се допълва с нулеви стойности, така че да стане със същия размер като по-големия и след това се извършва действието на оператора.

Задача 6.2. Имплементирайте булев вектор (съдържа само булеви стойностти) с размерност 32, като той трябва да заема **точно** 4 байта. За него имплементирайте следните методи:

- Оператор [] който се използва за четене на някой от компонентите по индекс (0-31), при достъпване на грешен индекс на компонент се връща -1.
- Метод bool set(int) сменя стойността на компонента на позиция, подадения аргумент, на 1, при викане с грешен индекс вектора не се променя и метода връща false.
- Метод bool clear(int) сменя стойността на компонента на позиция, подадения аргумент, на 0, при викане с грешен индекс вектора не се променя и метода връща false.

Напишете кратка main функция която демонстрира използването на 2-та вектора.

Задача 7. Да се реализира клас **войник**. Той трябва да съдържа информация за: име, години, ниво на бойните умения (цяло число) и заплата в златни монети.

- Да се реализира клас **сержант**, който наследява войник, но като допълнение има описание на взвода (от войници), който командва и масив от всички войници в него.
- Да се реализира клас **заклинание**, който съдържа неговото описание, тип и каква част от магическа мощ на магът, който го използва ,ще изисква.
- Да се реализира клас **магическа книга,** която съдържа страници със заклинания, всяко от които може да се използва точно веднъж (след това изчезва, заедно с цялата страница). Понеже книгата не е каква да е книга, а магическа – тя *винаги* има една празна страница накрая,давайки възможността на своя притежател да дописва ново заклинание.
- Да се реализира клас **маг**, който наследява войник и като допълнение съдържа описание на батальона (от взводове), който командва, масив от всички сержанти на взводовете, които са в този батальон, магическата мощ, която притежава и магическа книга.
- Да се реализира клас **главнокомандващ**, който наследява войник, има описание на войската, която предвожда и масив от магове които командват батальоните му.
- Да се реализира програма, която дава възможност да се въведе информация за една войска и по указател към главнокомандващ да изчисли:
    - Средното и общото ниво на бойните умения на войската.
    - Средното и общото ниво на магическата мощ на войската.
    - Колко злато на месец ще му излиза поддръжката на тази войска.

Бонус : Да се изчисли максималният брой заклинания, които могат да бъдат използвани от маговете в една войска (ако има такива). Взима се предвид, че всеки маг може да използва само своята магическа книга и при всяко използване на заклинание от книгата, магическата мощ на мага намалява със стойността, която заклинанието изисква за да бъде произнесено.

Забележки :

- Описанията трябва да са текстове с произволна дължина.
- В магическата книга страниците са колкото броя на заклинанията + 1 (за празната страница).
- Един маг не може да използва заклинание, което изисква повече магическа мощ от неговата.
- Всеки от командващите (сержант, маг и главнокомандващ) трябва има по-високо ниво на бойните умения от войниците (в частност това могат да са войници, сержанти и магове), които са под негово командване.
- Ако при реализацията използвате някакви библиотеки, то трябва да можете да обясните точно какво правят, защо ги използвате, както и да реализирате на място функциите които са в тези библиотеки, спазвайки характеристиките на библиотеката. Изключение се допуска за библиотеките за вход / изход (iostream, fstream), както и за cassert. За тях ще е нужно да обясните само каква е целта им и какви са добрите практики при използването им.
- Имате право да дефинирате допълнителни полета в тези класове, и всякакви методи, които ще ви помогнат да решите по-добре задачата си. Също така можете да дефинирате допълнителни класове.

Задача 8. Да се реализира клас *служител*. Той трябва да съдържа информация за име, адрес, ЕГН, отдел в който работи служителят и заплата му.

1. Да се реализира клас *специалист*, който наследява служител, но притежава като допълнение описание на специалността си.
2. Да се реализира клас *ръководител на отдел*, който наследява специалист и притежава информация на отдела, който ръководи, и указатели към всички служители в този отдел.
3. Да се реализира клас *секретарка*, за която имаме списък от езици които говори.
4. Да се реализира клас *директор*, който да има секретарка и указатели към всички служители във фирмата.
5. Да се реализира програма, която да дава възможност да се въведе информация за една фирма. По указател към директор на дадена фирма да се изчислява необходимата за заплатите сума.

Забележка

Имаме даден масив от служители. Дадена ни е и информация за необходимите за дадена фирма служители – по колко служители с дадена специалност са ни необходими, броят на отделите и специалностите им, броят на нужните служители неспециалисти. Да се определи възможно ли е от дадените служители да се състави фирма с исканите параметри, и ако може да се избере такъв екип, че заплатите да са минимални.

Задача 9. Да се реализира клас *регистрационен номер*. Класът трябва да представя номер на кола регистрирана в България. Изберете подходящ начин да съхраните номера на колата в класа. Реализирайте следната функционалност в класа:

- **Проверка за регион.** Класът трябва да има функция, която връща (като символен низ), името на региона, в който е регистрирана колата. За справка вижте статията https://bg.wikipedia.org/wiki/Регистрационен_номер_на_МПС_(България). Обърнете внимание, че всички допустими букви в номера, макар и да се произнасят на български, съответстват и могат да се представят като букви на латиница. Също, за улеснение можете да върнете името на региона, изписан на латиница или на английски.
- **Проверка дали номерът е обикновен или специален** (военен, дипломатически и т.н.). Информация за това има в същата статия.

Реализирайте клас *човек*. Класът трябва да може да пази информация за неговото име, ЕГН и телефонен номер.

Реализирайте клас *автомобил*. Автомобилът трябва да има марка, модел, година на производство, регистрационен номер, собственик и списък на лица, които имат разрешение да карат автомобила (шофьори). За регистрационния номер, собственика и лицата използвайте по-рано написаните от вас класове. Когато създавате техни обекти в рамките на класа за автомобил, в кои случаи това е агрегация и в кои - композиция?

Задача 10. Задачата ви е да реализирате клас - **текстов процесор**. Той се създава върху символен низ (const char *) и върху този низ могат да се прилагат множество **текстови трансформации**. В настоящата версия трябва да реализирате следните трансформации:

- преобразуване на текста към главни букви;
- преобразуване на текста към малки букви;
- замяна на всяко срещане на символ в текста с друг символ;
- замяна на всяко срещане на дадена последователност от символи в текста с друга дадена последователност.

Вашият текстов процесор трябва може да предостави както началния текст, така и този, получен след последователно прилагане на трансформациите. Трябва да поддържа също следните възможности и оператори:

- създаване по символен низ;
- създаване по символен низ и последователност от (нула или повече) трансформации;
- добавяне на нов текст към края на началния (преди прилагане на трансформациите) чрез предефиниран оператор;
- добавяне на нова трансформация чрез предефиниран оператор;
- извличане на символ по позиция (индексиране) от началния текст чрез предефиниран оператор;
- изход в поток на преобразувания текст чрез предефиниран оператор;
- премахване на трансформация от последователността по избран от вас критерий, чрез предефиниран оператор;
- проверка за равенство на два низа, след прилагане на трансформациите, чрез предефиниран оператор;
- възможност за включване / изключване на всички трансформации, чрез предефиниран унарен оператор –
- Бонус -> извличане на символ по позиция (индексиране) от трансформирания текст чрез предефиниран оператор;

Към реализацията са наложени следните изисквания и ограничения:

- Следвайте добрите ООП практики за реализация на класове и йерархии от тях, както и приетите добри практики при реализация на предефинирани операторни функции.
- Освен класовете изграждащи системата реализирайте и main функция, в която да демонстрирате употребата им.
- При разработка на класовете можете (и е задължително) да добавите методи, които са необходими за правилното решение на задачата, но не са явно посочени в условието.
- Всички масиви и символни последователности са с произволна дължина.
- Нямате право да използвате нищо от стандартните библиотеки, освен iostream.
- Код, който не се компилира или грубо нарушава принципите на ООП, ще бъде оценен с нула точки.

Задача 11. Да се реализира полиморфична йерархия от класове представляващи триизмерни фигури и да се реализира следната функционалност:

· По подадена точка да се извеждат фигурите които съдържат дадената точка

· По индекс на фигура [0, броя - 1] да се изведат нейните абсолютни координати (ако е в група се прилагат транслация и скалиране)

Фигурите които ще трябва да имплементирате са

1. Сфера - зададена по координатите на центъра и дължина на радиуса, пример:

sphere 3.0 7.5 6.3 3.15

2. Пирамида - зададен по координати на 4-те върха, пример:

pyramid 1.0 1.0 1.0 2.0 1.0 4.0 1.0 2.0 6.0 -2.0 0.5 4.3

3. Паралелепипед - зададен по 2 срещуположни върха, като неговите страни са успоредни на координатните оси, пример:

cuboid -1.0 -1.0 -1.0 2.0 2.0 2.0

4. Група - това представлява последователност от фигури (сфера, пирамида, паралепипед) които са транслирани с вектор и скалирани по някакъв скалар, пример с транслационен вектор (5.0, -7.3, 6.0) и скалар 4.0:

group in 5.0 -7.3 6.0 4.0 .... group out

Входа на програмата ще се чете от текстов файл, като всяка фигура ще бъде на отделен ред. Изключение прави групата, като тя ще се дефинира като започва от group in и завършва на group out, и всички редове между тези двата са фигури принадлежащи в тази група. Файлът винаги ще е правилен и не се налага да проверявате за коректност, няма да има случай на група която се съдържа в друга група.

Примерен файл:

```
sphere 3.0 7.5 6.3 3.15
pyramid 1.0 1.0 1.0 2.0 1.0 4.0 1.0 2.0 6.0 -2.0 0.5 4.3
group in 2.0 -1.0 1.0 2.0
cuboid -1.0 -1.0 -1.0 2.0 2.0 2.0
group out
```

За заявка на фигура на индекс 2 се извежда следното cuboid 0.0 -3.0 -1.0 6.0 3.0 5.0 като координатите са получени при скалиране на координатите на паралелепипеда и сбора им с транслиращият вектор от групата.

Забележки :

1. Една група съдържа точка в себе си, ако поне една от фигурите в нея съдържа точката (фигурите се разглеждат след скалирането и транслацията).
2. Ако на даден индекс има група на екрана трябва да се извеждат всички фигури участващи в групата.

Задача 12. Да се реализира клас Sorted<T>, който реализира "винаги сортиран" масив от елементи T. В Sorted<T> може да има максимум 128 елемента от тип T.
Класът да има методи add(T& elem), remove (T& elem) и print(), които съответно добавят елемент, махат елемент и отпечатват елементите разделени с нов ред.

Задача 13. Да се реализира клас EventLog, който съдържа списък от събития (описани от низове). Класът да предоставя възможност за регистриране на ново събитие и да има възможност да регистрира произволен брой събития. Да се реализират подходящи конструктори, деструктори и оператори за този клас, както и метод AddEvent, който регистрира събитие. Обърнете внимание, че е добре да пазите копие на текста на събитието вместо указателя, подаден от потребителя. Да се реализира и метод Print, който отпечатва на екрана всички събития.
Забележка. Не е нужно събитията да могат да бъдат редактирани и премахвани!

Задача 14. Да се реализира клас Building, който описва дадена сграда с височина (цяло число метри), площ (число с плаваща запетая в кв.м) и адрес (низ с произволна дължина). Да се реализира производен клас House, който допълнително задава брой етажи (цяло число) и име на собственик (низ с произволна дължина). За класовете да се реализират:

а) подходящи селектори и мутатори;

б) функции за въвеждане от клавиатура и извеждане на екрана;

в) функция, която по даден масив от къщи намира най-просторната къща, т.е. тази с най-голяма средна височина на етаж.

Задача 15. Да се реализира абстрактен базов клас IntSet, който описва следните операции върху изброимо крайно множество от цели числа:

- bool member(int x): проверява дали цялото число x е елемент на множество.
- int get(int i): връща i-тия елемент на множество. Индексацията на елементите е без значение.
- bool operator < ([попълнете правилния тип] s): проверява дали дадено множество е същинско подмножество на s.
- bool operator * ([попълнете правилния тип] s): проверява дали две множества имат непразно сечение.

Да се реализират наследници IntRange и ArraySet. Клас IntRange представя затворен интервал от цели числа. Краищата на интервала да се задават при конструиране на обекта. ArraySet е клас, поддържащ множество от максимум n цели числа, където n се задава по време на конструиране на обекта. Класът да поддържа следните операции:

- bool insert(int x): добавя числото x към множеството. Ако капацитетът е изчерпан, връща лъжа. Връща истина в противен слуай.
- bool remove(int x): премахва числото x от множеството. Ако елементът не съществува, резултатът е лъжа. Резултатът е истина в противен случай.

Да се реализира функция bool mon([попълнете правилния тип] sets[], int n), която получава

масив от обекти (или указатели към обекти) множества. Масивът е с големина n. Функцията да проверява дали елементите на масива образуват строго монотонно растяща редица.

Задача 16. Да се реализират:

- структура **Company**, описваща компания с име до 100 символа и БУЛСТАТ (9-цифрен номер);
- шаблон на клас за физическо лице **Person<T>**, който съдържа:
  - име (низ с произволна дължина);
  - ЕГН (10-цифрено число)
  - осигурител (указател към обект от тип T);
  - подходящ конструктор;
  - голяма четворка (осигурителят не се копира, само указателя към него);
- клас **Employee**, производен на **Person** със **Company** като осигурител, който съдържа:
  - допълнителна член-данна, описваща длъжността на служителя (низ с произволна дължина);
  - подходящ конструктор;
  - голяма четворка;
  - член-функция за извеждане на информация за служител и неговия осигурител.

Забележка: реализирайте методите от голямата четворка само когато се налага, т.е. когато системно генерираната голяма четворка не върши работа.

Задача 17. Да се реализират:

- абстрактен базов клас **Device**, описващ техническо устройство, който съдържа производител (низ с произволна дължина) и поддържа две операции:
  - **print:** за извеждане на информация за устройство;
  - **get_perf:** за намиране на числова мярка за производителност на устройството;
- производен клас **Laptop**, описващ лаптоп, който има допълнителна член-данна за скорост (цяло число мегахерци). Да се дефинират двете операции **print** и **get_perf**, като:
  - информацията за лаптопа се състои от неговия производител и скорост;
  - мярката за производителност на лаптопа е неговата скорост;
- производен клас **Car**, описващ автомобил, който има допълнителни член-данни за мощност (цяло число киловати) и обем на двигателя (цяло число cm^3). Да се дефинират двете операции **print** и **get_perf**, като:
  - информацията за автомобила се състои от неговия производител, мощност и обем;
  - мярката за производителност на автомобила е неговата мощност;
- клас **Inventory**, описващ фирмен инвентар от устройства (до 100 на брой), които могат да бъдат лаптопи и коли. Да се реализират:

- функция, която извежда информация за всички устройства в списъка;
- функция, която проверява дали инвентарът е подреден в нарастващ ред по производителност.

Забележка: реализирайте методите от голямата четворка само когато се налага, т.е. когато системно генерираната голяма четворка не върши работа.

Задача 18. Да се реализира абстрактен базов клас `Question`, който описва следните операции върху въпрос от тест:

- `ask`, която извежда въпроса и въвежда негов отговор;
- `grade`, която оценява въпрос и връща броя точки.

Да се реализират следните видове въпроси:

- `YesNoQuestion`, който описва въпрос с два възможни отговора: да или не. При конструиране се задават текст, точки и верен отговор. Въпросът дава 0 т. при грешен и пълен брой точки при верен отговор.
- `MultipleChoice`, който описва въпрос с няколко възможни отговори, от които някои са верни. При конструиране се задават текст, точки **x**, възможни и верни отговори. Всеки правилно посочен отговор добавя **x** точки, а всеки погрешно посочен отговор отнема **x** точки.
- `OpenQuestion`, който описва въпрос със свободен текст. При конструиране се задават текст и брой точки. При оценяване се извеждат въпроса и отговора и се въвежда оценка в проценти. Ако отговор не е даден, въпросът автоматично се оценява с 0 т.

Да се реализира клас `Test`, представящ тест с въпроси от всякакъв вид. Да се реализират функциите:

- `addQuestion`, която добавя нов въпрос към теста;
- `doTest`, която задава всички въпроси подред и събира отговорите им;
- `gradeTest`, която оценява теста и връща броя събрани точки.

Бонус: реализирайте метод `void shuffle()`, който разбърква въпросите в даден тест на произволен принцип. Използвайте библиотечната функция `int rand()`, която връща случайно цяло число.

Задача 19. Да се дефинира шаблон на клас `Relation<T>`, който съдържа два обекта от тип T, наречени `subject` и `object`, и низ с произволна дължина `relation`, описващ връзката между тези обекти.

Пример: `Relation<int> r1(2,6,"is smaller than"),r2(6,3,"is divisible by");`

За шаблона да се реализират голямата четворка и операция за отпечатване void print()

Пример: `r1.print(): 2 is smaller than 6.`

За инстанцията на шаблона Relation<int> реализирайте и оператор за композиция * по следния начин.

Ако `r = r1 * r2`, то `r.subject = r1.subject, r.object = r2.object, r.action = r1.action r1.object ", which is" r2.action`

Пример: `(r1*r2).print(): 2 is smaller than 6, which is divisible by 3`

Композицията се допуска само ако `r1.object == r2.subject`, в противен случай резултатът е `r1`.

Задача 20. Да се реализира шаблон на абстрактен базов клас `Container<T>`, дефиниращ следните методи за работа с контейнери:

- `bool member (const T&) const` : проверка за принадлежност на елемент
- `bool add (const T&)`: добавяне на елемент към контейнера, върнатата стойност показва дали добавянето е било успешно
- `bool remove (const T&)`: премахване на (първото срещане на) елемент от контейнера, върнатата стойност показва дали премахването е било успешно
- `int size () const`: брой елементи в контейнера
- `T& operator [] (int i)`: достъп до i-тия елемент на контейнера

а) Да се реализира шаблон на клас `LinkedVector<T>`, наследник на `Container<T>`, реализиращ контейнер с неограничен капацитет чрез линеен едносвързан списък, позволяващ по няколко екземпляра от една и съща стойност.

б) Да се реализира шаблон на клас `ArrayVector<T>`, наследник на `Container<T>`, реализиращ контейнер с ограничен капацитет чрез динамичен масив, позволяващ по няколко екземпляра от една и съща стойност. Капацитетът на контейнера да се задава по време на конструиране, като по подразбиране да е 10 елемента.

в) Да се реализира шаблон на клас `Set<T>`, наследник на `Container<T>`, реализиращ контейнер, който позволява най-много по един екземпляр от всяка стойност (например, числото 5 може да се среща най-много един път в множество от числа). Шаблонът `Set<T>` да съдържа член-данна, реферираща обект `store` от клас, наследник на `Container<T>`, който отговаря за съхранението на стойностите от множеството. Обектът `store` да се задава като параметър на конструкторът на `Set<T>`. Методите на `Set<T>` да се реализират така, че да "делегират" на `store` съхранението на обектите, като го използват по такъв начин, че да се осигури липсата на повторение на стойности.

г) Да се реализира функция

[подходящ тип] `intersection (`[подходящи параметри]`)`,

която по масив от произволни контейнери, построява и връща множеството от общите им елементи.

Задача 21. а) Да се дефинира клас **Walker**, описващ "подвижен обект", който изминава определено разстояние за определено време. Обектите да се характеризират със следните атрибути:

- описание (символен низ с произволна дължина)
- средна постижима скорост в km/h (число с плаваща запетая)

Да се реализира метод **double walk (double distance),** който пресмята за колко часа обектът изминава определено растояние (на базата на средната му постижима скорост).

б) Да се дефинира клас **Vehicle**, произведен на **Walker**, който има допълнителната

характеристика **amortisation** (число с плаваща запетая), задаващо брой километри, изминати от обекта до настоящия момент. При всяко изпълнение на **walk**, **amortisation** да се актуализира с изминатото разстояние. Формулата, изчисляваща времето за изминаване на растоянието да взема предвид, че след всеки 100 изминати километра, средната постижима скорост намалява с 1 km/h, но не става по-ниска от 50% от първоначалната стойност на средната постижима скорост за обекта. Да се реализират всички необходими конструктори на класа **Vehicle**.

в) Да се реализира клас **Fleet**, който да поддържа списък от не повече от 15 подвижни обекта. Позволено е използването на шаблона **Array** от Задача 1 или на друго средство за поддържане на последователност от обекти. Класът да поддържа следните методи:

- **void addWalker ([подходящ тип])**, който добавя нов подвижен обект към флотилията. Обектът може да е както от тип **Walker**, така и от тип **Vehicle**;
- **double walkAll (double distance)**, който дава средната скорост, която цялата флотилия развива при едновременното изминаване на **distance** километра от всеки от обектите, като се взема предвид натрупваната във флотилията амортизация.

Задача 22. Да се реализира шаблон на абстрактен базов клас `MathExpression<T>`, описващ математически израз от тип `T`. Класът да дефинира метод `T value() const`, намиращ числовата стойност на израза и метод `void print() const`, отпечатващ израза на екрана.

а) Да се дефинира реализация `Constant<T>`, която представя константа от тип `T`. Стойността на константата да се задава при конструиране на обекта.

б) Да се дефинира реализация `Max<T>`, която представя максимума на най-много 10 произволни израза от тип `T`. Класът да реализира метод за добавяне на израз, като се предполага, че броят на добавените изрази няма да надхвърля 10. При конструиране на обекти се смята, че те не съдържат никакви изрази и в този случай максимумът по дефиниция е 0.

в) Да се дефинира реализация `Average<T>`, която представя средното аритметично на най-много 10 произволни израза от тип `T`. Класът да реализира метод за добавяне на израз, като се предполага, че броят на добавените изрази няма да надхвърля 10. При конструиране на обекти се смята, че те не съдържат никакви изрази и в този случай средното аритметично по дефиниция е 0.

г) Да се дефинира реализация `Sum<T>`, която представя сумата на два произволни израза. Самите изрази да се подават по подходящ начин при конструиране на обекти от клас `Sum<T>`.

д) Да се дефинира реализация `Prod<T>`, която представя произведението на два произволни израза. Самите изрази да се подават по подходящ начин при конструиране на обекти от клас `Prod<T>`.

Задача 23. Да се реализира клас **Task**, който представя задача, характеризираща с име (низ до 100 символа) и време за изпълнение (цяло число часове). Да се реализира клас **RecurringTask**, който представя периодична задача, която освен горните характеристики има и цяло число, задаващо брой повторения. За дадените класове да се реализират:

а) подходящи конструктори;

б) метод `getTotalTime`, който връща общото време за изпълнение на задачата. Общото време за изпълнение на периодична задача се получава като произведение на времето за изпълнение и броя повторения;

в) метод `print`, който извежда на стандартния изход информация за задачата;

г) метод `printShorter`, който по подаден масив от задачи извежда на стандартния изход информация за тези от тях, които се изпълняват за по-малко време, отколкото задачата, за която се вика метода. В масива може да има задачи и от двата вида: обикновени и периодични.

Задача 24.

а) Да се реализира абстрактен клас **Issue**, предсталяващ периодично издание. Всяко периодично издание се характеризира със заглавие (низ с произволна дължина), цена за 1 брой (положително дробно число) и брой издания за година (положително цяло число).

б) За класа **Issue** да се дефинират голямата четворка, подходящ конструктор с параметри, както и подходящи селектори и мутатори.

в) Да се реализира клас **Subscription**, който представлява годишен абонамент за едно или повече периодични издания, описани чрез масив

г) За класа **Subscription** да се дефинират голямата четворка, подходящ конструктор с параметри, както и подходящи селектори и мутатори.

д) Да се дефинира `operator+=` , който добавя издание в масива

е) Да се дефинира `operator[]` , който връща заглавието на периодичното издание на съответния индекс в масива, ако такова издание съществува.

ж) Да се дефинира метод `calculatePrice`, който изчислява общата цена за годишен абонамент на всички издания в масива, с 30% отстъпка за всяко от изданията, като взема предвид броя издания за година.

Задача 25.

1. Да се реализира клас **Song**, който описва музикално изпълнение със следните свойства:
    - заглавие (низ до 100 символа)
    - изпълнител (низ до 100 символа)
    - година на издаване (цяло число)
    - дължина на песента (цяло число секунди)
    - оригинал (ако изпълнението е “кавър”, указател към оригиналната песен)
2. За класа **Song** да се реализират:
    - подходящи конструктори, селектори и мутатори
    - голяма четворка, ако е нужна
    - операция за извеждане на информация за песен на стандартния изход

3. Клас **Album**, който описва албум със следните полета:
    - списък от песни с произволна дължина
    - име на албума (низ до 100 символа)
4. За класа **Album** да се реализират
    - подходящи конструктори, селектори и мутатори
    - голяма четворка, ако е нужна
    - операция за извеждане на информация за албум на стандартния изход
    - операция **getLength**, която намира дължината на албума в секунди
    - операция **getArtist**, която връща името на изпълнителя на албума, ако всички песни са на един и същ изпълнител, или “Various Artists”, ако албумът се състои от песни на повече от един изпълнител
    - операция **findSong**, която връща указател към първата песен в албума с дадено име, или NULL, ако такава не е намерена
    - операция **deleteDuplicates**, която изтрива от албума всички дублиращи се песни. Две песни се считат за дублиращи се, ако за тях съвпадат заглавието, изпълнителя, годината на издаване и дължината
    - операция **detectCovers**, която засича “кавърите” в албума и насочва техните указатели към съответните оригиналните песни. Считаме, че една песен A е “кавър” на друга песен B, ако:
        i. A и B имат едно и също заглавие, но различен изпълнител
        ii. A е издадена след B и в албума няма песен, която да отговаря на предното условие и е по-рано издадена от B
5. Да се реализира примерна програма, която демонстрира всички гореизброени операции

Задача 26. Да се реализира клас **Course**, който описва разписанието на лекциите на курс във ФМИ през даден семестър със следните полета:

- име на курса (низ с произволна дължина)
- специалност (низ до 30 символа, като празен низ означава, че курсът е избираем)
- година (число от 0 до 4, като 0 означава избираем курс)
- поток (число от 0 до 2, като 0 означава избираем курс)
- ден от седмицата (число от 1 до 6)
- начален час (число от 7 до 20) и продължителност (число от 1 до 6), като курсът не може да завършва по-късно от 21 часа
- аудитория (число от 1 до 599)

За класа да се реализират подходящи конструктори, селектори и мутатори и операции за вход (>>) и изход (<<).

Да се реализира клас **SemesterSchedule**, който представя програмата за даден семестър във ФМИ като съдържа:

- масив с произволна дължина от обекти от клас **Course**,
- масив от числа, описващ наличните във ФМИ аудиторни зали.

За класа да се реализират следните операции:

- операция **addCourse** за добавяне на нов курс към програмата. Операцията е успешна само ако:
    - курсът задава коректна аудиторна зала,
      И
    - в програмата няма друг курс, който е:
        - по същото време в същата зала,
          ИЛИ
        - по същото време и е обявен за същия поток, курс и специалност, като изключение правят избираемите курсове.
    - в случай на конфликт, на стандартния изход за грешка (cerr) да се извежда информация за курса, с който се получава конфликт
- операция **printAuditoriumSchedule**, която извежда програмата по дни за дадена аудитория
- операция **printClassSchedule**, която извежда програмата по дни за даден поток на даден курс на дадена специалност. При подходящи стойности на параметрите да се извежда програмата на избираемите курсове.
- операция **findFreeSlot**, която добавя в програмата даден курс, като автоматично намира време и зала, в която може да се проведе курса. Операцията да се съобразява с това да не се получава конфликт с вече съществуващ курс. Операцията е неуспешна, ако не могат да се намерят подходящи време и зала за дадения курс.

Задача 27. Да се реализира клас **String**, който представя низ (поредица от символи) чрез верига от двойни кутии. Например, низът “Hello” да се представя по следния начин:

За класа да се реализират подходящи функции, така че да са допустими следните операции:

- `String eps;` // дефинира празен низ
- `String s = “Hello”;` // дефинира низ със стойност “Hello”
- `String s2 = s;` // дефинира s2 като копие на s
- `String s3; s3 = s;` // дефинира празен s3 и копира s в него
- `cin >> s;` // въвежда низа s от стандартния вход
- `cout << s;` // извежда низа s на стандартния изход
- `s = “Hello”;` // s получава стойност “Hello”
- `s[4]` // връща символа на 4-а позиция, ‘o’
- `s[1] = ‘a’;` // променя втората буква на s, така че
  // s става “Hallo”, връща резултат s[1]
- `s + s2` // конкатенира s и s2, резултат “HalloHello”
- `s + “, world!”` // резултат “Hallo, world!
- `“Friend, “ + s` // резултат “Friend, Hallo”

```
'#' + s          // резултат "#Hallo"
s + '!'          // резултат "Hallo!"
s += '!'         // s става "Hallo!", връща резултат s
s += "?!"        // s става "Hallo!?!", връща резултат s
s * 3            // връща "Hallo!?!Hallo!?!Hallo!?!"
s *= 3           // s става "Hallo!?!Hallo!?!Hallo!?!"
s2 == s3         // проверява дали s2 и s3 са равни (true)
s != s2          // проверява дали s и s2 са различни (true)
s < s2           // проверява дали s е преди s2 (true)
s2 < s3          // проверява дали s2 е преди s3 (false)
s + 'a' >= s     // проверява дали s + 'a' не е преди s (true)
s <= s + 'b'     // проверява дали s не е след s + 'b' (true)
s['?']           // намира първото срещане на '?' в s (6)
s['*']           // ако няма такъв символ, връща -1
s["llo"]         // намира първото срещане на "llo" в s (2)
s["ell"]         // ако няма такъв подниз, връща -1
s()              // връща дължината на s (24)
s(18)            // връща подниз на s от 18-тата позиция до края
                 // резултат "llo!?!"
s(4,8)           // връща подниз на s от 4-а до 8-а позиция
                 // включително, резултат "o!?!H"
```

Задача 28. Контакти

Първа част: Речник

Да се реализира шаблон на клас `Dictionary` с два типови параметъра K и V. Класът да представя речник, който съдържа редица от ключове от тип K и на всеки ключ съпоставя по една стойност от тип V. Едно примерно представяне би могло да бъде с два масива с една и съща дължина — един с елементи от тип K и един с елементи от тип V. За типа K и V може да се приеме, че са налични:

- голяма четворка
- операция за изход <<
- операции за сравнение == и !=
- за типа K допълнително са налични операции за наредба <, <=, >, >=

За класа да се реализират:

- голяма четворка
- добавяне на ключ и съответна на него стойност
- търсене на стойност по ключ
- извежда речника на стандартния вход подреден по ключ
- разширяване на речника при изчерпване на капацитета му

Втора част: Книга с контакти

Да се създаде клас `Contact`, който съдържа име, телефонен номер (до 10 цифрен), и идентификатор с подходящи конструктор, селектори и мутатори.

Да се създаде клас `ContactBook`, който представя бележник с контакти като набор от три речника, чиито ключове са съответно име, телефонен номер и идентификатор, а стойностите са указатели към контакти (`Contact*`).

За този клас да се реализират:

- добавяне на контакт
- намиране на контакт по даден критерий (име, телефонен номер или идентификатор). *Внимание: върнатият контакт да няма възможност да бъде променян!*
- извеждане на книгата с контакти подредена по даден критерий (име, телефонен номер или идентификатор)
- премахване на контакт

Упътване 1: Може да се използва наготово класът от първото домашно `String` или стандартния клас `std::string`.
Упътване 2: Може да се предефинира операцията за изход << за тип `Contact*`, за да работи правилно извеждането на речник.

Забележка: за тази част от задачата не се изискват никакви усилия по управление на динамичната памет.

Трета част: Динамична памет

За класа `ContactBook` да се реализира допълнителна логика, която да се грижи правилно за управлението на подаваните контакти. Считаме, че `ContactBook` “притежава” контактите, т.е. при добавяне на контакт създава негово копие, а при изтриване унищожава контакта.

Задача 29. Формули

Част първа: Йерархия от класове

Да се дефинира абстрактен базов клас `Formula`, дефиниращ операцията `double value() const`. Класът да представя формула с неуточнено представяне, чиято стойност може да бъде изчислена чрез метода `value`. Да се дефинира също и операция `void print() const` за извеждане на явния вид на формула на стандартния изход.

а) Да се дефинира производен клас `Constant`, който описва дробни константи. Обекти от клас `Constant` се конструират със стойността на съответната константа.

Пример:

```
Constant a (0), b (1);
cout << b.value();    // извежда 1
b.print();            // също извежда 1
```

б) Да се дефинира клас `BinaryOperation`, описващ приложение на двуместна аритметична операция върху други две формули. Допустимите операции са `+`, `-`, `*`, `/`, както и операциите за сравнение `<` и `=`, като последните имат стойност 1 при удовлетворено (не)равенство и стойност 0 иначе. Обекти от клас `BinaryOperation` се конструират със символ, описващ операцията и два указателя към формули, задаващи операндите.

Пример:

```
BinaryOperation test ('*',
            new Constant (2),
            new BinaryOperation ('+',
                      new Constant (1),
                      new Constant (3));

cout << test.value(); // извежда 8, т.е. стойността на 2 * (1 + 3)
test.print();         // извежда "( 2 * ( 1 + 3 ) )"
```

в) Да се дефинира клас `Read`, описващ стойност, въведена от потребителя.
Пример:

```
BinaryOperation test ('*', new Constant (2), new Read);
cout << test.value();  // изисква от потребителя да въведе
                       // число и го връща умножено по 2
test.print();          // извежда "( 2 * Read )"
```

г) Да се дефинира клас `Conditional`, описващ формула, която зависи от дадено условие. Обекти от клас `Conditional` се конструират с три указателя към `Formula` — условие (“if”), формула при удовлтеворяване на условието (“then”) и формула при неудовлетворяване на условието (“else”).

Пример:

```
Conditional test (new BinaryOperation ('<', new Read,
                                            new Constant (5)),
              new Constant (1),
              new Constant (2));

cout << test.value(); // извежда 1, ако поребителят въведе
```

```
                        // число, по-малко от 5
                        // и 2 в противен случай

test.print(); // извежда "if ( Read < 5 ) then 1 else 2"
```

Забележка: за тази част от задачата не се изискват никакви усилия по управление на динамичната памет.

Втора част: Управление на динамичната памет

За така изградената йерархия да се дефинират необходимите специални методи (конструктори, деструктори, оператор за присвояване) така, че динамичната памет да се управлява коректно. Следните операции трябва да могат да протичат без проблеми, свързвани със споделяне на памет, изтичане на памет или други неправилни манипулации на динамичната памет. След приключването на съответния им блок, заетата динамична памет трябва да бъде освободена.

```
Constant c1 (1);
Constant* c2 = new Constant (2);
BinaryOperation f1 ('+', &c1, c2);
BinaryOperation f2 = f1;   // копиране
f2 = f1;         // присвояване
delete c2;
```

Упътване 1: В класа `Formula` да се добави виртуален метод `Formula* clone() const`. Всеки клас да предефинира метода така, че обектите да могат да построяват свои идентични копия.
Упътване 2: Да се променят класовете така `BinaryOperation` и `Conditional`, така че да "притежават" формулите, които ги съставят. В частност конструкторите да бъдат построени така, че да запомнят в член-данните адресите на копия на параметрите си, а не оригинално подадените адреси. Дестурктурите, от своя страна, да изтриват копията, записани в съответните член данни.

Трета част: Десериализация (4 т.)

Показаните в примерите резултати от метода `print` за всеки клас от описаната йерархия демонстрират едно еднозначно представяне на всички формули, представими с нея. Да се дефинира рекурсивна функция `Formula* readFormula ()`, която по въведено от стандартния вход правилно построено представяне на формула построява съответното ѝ представяне чрез класовете от изградената йерархия. Пример:

```
Formula* f = readFormula(); //от стандартния вход се въвежда
                            // "if ( Read < 5 ) then 1 else 2"
```

```
cout << f->value(); // извежда 1, ако поребителят въведе
                    // число, по-малко от 5
                    // и 2 в противен случай

delete f;
```

Забележка: По ваше желание можете да изберете друго, по-просто представяне на формула.

Задача 30. Да се реализира клас `Cashier`, който описва каса в магазин с име на касиера (низ с дължина до 30 символа) и оборот (дробно число лева). Да се реализира клас `Store`, който описва магазин с до 10 каси. За двата класа да се реализират:

- подходящи конструктори и голяма четворка, ако е нужна;
- подходящи селектори и мутатори;
- операция `>>` за въвеждане от стандартен вход;
- операция `<<` за извеждане на стандартен изход.

Допълнително, за клас `Store` да се реализират следните функции:

- метод, който намира общия оборот за магазина;
- операция `[]`, която връща каса в магазина по зададено име на касиер;
- операции за сравнение, които сравняват два магазина по "доходност", като за по-доходен се счита този магазин, за който средният доход на каса е е по-голям.

Забележка. Не е нужно да се реализират мутатори за свойства, които не е смислено да бъдат променяни.

Задача 31.

а) Да се създаде клас **Book**,който да съдържа следните данни,характерни за една книга:
- заглавие на книгата
- брой страници

Да се използва динамична памет за символните низове.Да се създадат необходимите селектори и мутатори,както и голяма четворка + конструктор с подадени параметри.
Предефинирайте оператори:
- за изход (<<)
- за вход (>>)
- оператори > и < (сравнява книги лексикографски по заглавие)

Hint: за оператори > и < използвайте strcmp

б) Да се създаде клас **Library**, който да съдържа динамичен масив от книги.Да има възможност за добавяне и махане на книга от масива, както и намиране на средното аритметично на броя страници на всички книги в масива. Предефинирайте оператор [],

който да връща книга по поредния ѝ номер в масива. Напишете функция sort_by_title(char const* title) ,която да сортира масива от книги по заглавията им.

в) Да се създаде клас **EBook**(онлайн книга), който наследява класа **Book**.Като член-данна да се съдържа размера на книгата в байтове(int) и брой отваряния на книгата. Добавете необходимите мутатори, селектори и конструктор по зададени данни. Да се напише функция,която връща рейтинга на книгата чрез формулата:
- ако броя на отваряният е над 200,рейтинга е 5
- иначе рейтинга е брой на отваряния / 40

Задача 32. Да се реализират абстрактни базови класове `Input` и `Output`, който описва входно и изходно устройство и позволяват работа с цели числа. Входните устройства поддържат следните операции:

- `read`, която прочита число от входното устройство;
- `canRead`, която проверява дали могат да се въведат още числа.

Изходните устройства поддържат следните операции:

- `print`, която извежда число на изходното устройство;
- `canPrint`, която проверява дали могат да се изведат още числа.

Да бъдат реализирани следните устройства:

- `Reader`, входно устройство, което въвежда цели числа от стандартния вход, като спира при прочитане на числото 0;
- `PrintSum`, изходно устройство, което натрупва изведените на него числа и извежда текущата сума, като спира приемане на нови числа при получаване на сума по-голяма от 10000;
- `StackMemory`, входно-изходно устройство, представено от стек с фиксиран капацитет, като извеждането на числа ги добавя на върха на стека, а въвеждането ги премахва.

Дадени са два масива с еднаква дължина от входни и от изходни устройства. Да се напише функция `transfer`, която обхожда циклично устройствата, като въвежда по едно число от поредно входно устройство и го извежда на съответното му изходно устройство, ако двете операции са възможни, и след това преминава на следващата такава двойка устройства.

Задача 33. Да се реализира шаблон на клас `Counters<T>`, съдържащ два брояча, които помнят колко пъти е прочетена и колко пъти е променена стойността на данна от тип T. За шаблона да се реализират подходящи селектори, мутатори и операция `+=`, която добавя броячите на подаден обект от тип `Counters<T>`, но само ако данната от тип T съвпада (в смисъла на `==`). Да се реализира шаблон на клас `CounterArray<T>`, който представя масив от данни от тип T с броячи. За шаблона да се реализират функциите:

- операция `[]`, която дава достъп за четене до пореден елемент;
- `sort`, която сортира масив от елементи от тип T по метода на пряката селекция и извежда статистика за това кой елемент от масива колко пъти е бил използван;
- `statistics`, която използва операцията `+=`, за да изведе статистика за използването всеки уникален елемент от масива.

Задача 34. Да се реализира шаблон `Named<T>`, който позволява добавяне на име (низ с произволна дължина) към данна от произволен тип T. За шаблона да се реализират голяма четворка, подходящи селектор и мутатор и операции за въвеждане (`>>`) и извеждане (`<<`). Да се реализира подходящ наследник `Variable`, който описва целочислена променлива със име и стойност. За `Variable` да се реализират:

- едноместна операция `*`, която връща стойността на променливата;
- операция `=`, която присвоява стойността на една променлива на друга без да променя името;
- операции `==` и `!=`, които сравняват две променливи. Две променливи се считат равни, ако имената и стойностите им са равни.

Задача 35.

А) Да се дефинита клас Word, описващ дума, съставена от не повече от 20 символа от тип char. Класът да съдържа следните операции:

1. оператор [] за намиране на i-тия пореден символ в думата
2. оператори + и += за добавяне на един символ в края на думата. Ако думата вече има 20 символа, операторите да нямат ефект
3. оператори < и == за сравнение на думи спрямо лексикографската наредба
4. подходящи конструктори

Б) Де се дефинира клас Sentence, описващ символен низ, състоящ се от проиволен брой символи от тип char. Класът да поддържа следните операции

1. функция addWord за добавяне на дума (обект от клас Word) към края на изречението
2. конструктор за копиране
3. други подходящи контруктори
4. оператор за присвояване
5. деструктор
6. метод за извеждане в поток (print)

В) Да се дефинира клас EnglishSentence, наследник на клас Sentence, който допуска изречения, съставени единствено от малки и големи латински букви и символите интервал, запетая, удивителен знак и точка. Упътване: за целта да се дефинира подходящ вариант на метода addWord.

Г) Да се дефинира функция

void addAndPrint ([подходящ тип]sentences, [подходящ тип]words[,...]),

където sentences е масив от произволен брой разнородни изречения (едновременно обекти от класовете Sentence и EnglishSentence), а words е масив от произволен брой думи. Ако е необходимо, функцията да има и допълнителни параметри. Фунцкията добавя всяка от думите от words към всяко от изреченията в sentences и след това отпечатва всички така получени изречения.

Задача 36. Да се реализира шаблон на клас `BinaryRelation`, който описва двуместна релация между елементи от тип `T` и елементи от тип `U`. Релацията се представя като последователност от двойки, които са в релация. За шаблона да се реализират следните операции:

- подходящи конструктори
- `r(t,u)`, която проверява дали t и u са в релация
- `!r`, която намира обратната релация на `r`
  (всички двойки (u,t), които са в релация `r`)
- `r1 + r2`, която намира обединението на `r1` и `r2`
  (всички двойки (t,u), които са в релация r1 или в релация r2)
- `r1 ^ r2`, която намира сечението на `r1` и `r2`
  (всички двойки (t,u), които са в релация r1 и в релация r2)
- `r[t]`, която намира образа на елемента `t`
  (всички елементи u, за които (t,u) са в релация r)
- `r(u)`, която намира първообраза на елемента `u`
  (всички елементи t, за които (t,u) са в релация r)
- `r1 * r2`, която намира композицията на релациите `r1` и `r2`
  (всички двойки (t,v), за които има u, така че (t,u) е в r1 и (u,v) е в r2)
- `r.dom()`, която намира домейна на `r`
  (всички елементи t, които участват в някоя двойка (t,u) от r)
- `r.ran()`, която намира образа на `r`
  (всички елементи u, които участват в някоя двойка (t,u) от r)
- операциите за съкратено присвояване `+=`, `^=` и `*=`
- операции за вход (`>>`) и изход (`<<`), които записват релацията в поток с подходящ формат
- предикати
  - `function`, който проверява дали релацията е графика на частична функция
  - `injection`, който проверява дали релацията е графика на инективна функция

С помощта на шаблона `BinaryRelation` да се реализира шаблон `KnowledgeBase`, който описва “база от знания”: множество от връзки между двойки елементи от тип `T` и `U`, като всяка връзка има низ за етикет.

Пример: за `T = int`, `U = char const*`, една база от знания би могла да съдържа следните връзки (използван е форматът <име-на-връзка>(<елемент-от-тип-T>, <елемент-от-тип-U>)):

- `nameOf(4, "four")`
- `nameOf(42, "fourty-two")`

- `nameOf(42, "forty-two")`
- `nameOf(5, "five")`
- `nameOf(4, "vier")`
- `nameOf(99, "neunundneunzig")`
- `romanNumeral(5, "V")`
- `romanNumeral(42, "XLII")`
- `romanNumeral(42, "xlii")`
- `binaryString(4, "100")`
- `binaryString(5, "101")`
- `hexString(42, "2A")`
- `hexString(42, "2a")`
- `hexString(99, "63")`
- `octString(51, "63")`
- `octString(65, "101")`
- `famousfor(10, "number of fingers on human hand")`
- `famousfor(10, "smallest number with two decimal digits")`
- `famousfor(99, "largest number with two decimal digits")`
- `famousfor(42, "answer to the ultimate question of life, the universe, and everything")`

`KnowledgeBase` трябва да поддържа всички операции, които поддържа `BinaryRelation`, като изпълнява съответните операции едновременно върху всички релации в `KnowledgeBase`. В допълнение, `KnowledgeBase` да поддържа операцията `kb("relationName")`, която връща релацията с име `relationName` от базата от знания `kb`. Ако няма нито една връзка с това име, операцията да връща празната релация.
Да се реализира програма, която демонстрира действието на шаблоните с подходящи примери с различни стойности на типовите параметри `T` и `U`.

Задача 37. Да се реализира абстрактен базов клас `Rating`, който описва алгоритъм за класиране на списък от кандидати. Класът да поддържа следните операции:

- `dataSource(a, n)`, която задава списък от имена a с дължина n
- `top(n)`, която връща n-те най-добри кандидата според алгоритъма

Да се реализират следните алгоритми:

- `LengthRating`, който класира имената по дължина, от най-късо към най-дълго
- `BiasedRating`, който се конструира предварително със списък от имена на предпочитани кандидати и класира тях на първи позиции. Кандидатите, които не се срещат в списъка с предпочитания, се класират в първоначален ред
- `CombinedRating`, който комбинира два алгоритъма за класиране R1 и R2 по следния начин: ако едно име е на място i по R1 и на място j по R2, то той получава i+j точки. Кандидатите с по-малко точки се класират по-напред.

Да се реализира функция, която приема списък от кандидати, списък от алгоритми за класиране и име. Като резултат функцията да връща такъв алгоритъм, в който името е

класирано на някое от първите 5 места. Ако такъв няма, връща подходяща нова инстанция на `BiasedRating`, която удовлетворява това условие.

Задача 38. Да се реализира клас `ParkingMachine`, който представя паметта на автомат за паркиране. В паметта на автомата се пази списък с номерата на паркиралите коли (низ от 7 или 8 символа) и за всяка кола се пази минута на влизане в паркинга (броено от 0:00 часа). Освен това, при конструиране се задава цената на час паркинг като дробно число. Първоначално автоматът може да запази информация за 10 коли, но при препълване на паметта да удвоява капацитета си до достигане на лимита на паркинга от 1280 коли. За класа да се реализират:

- голямата четворка,
- метод `enter(regno, time)`, който записва информация за новопаркирала кола с номер `regno` във време `time`,
- метод `leave(regno, time)`, който изтрива информацията за кола с номер `regno` и връща сумата, която се дължи за паркиране, ако текущото време е `time`. Таксуването е на всеки започнат час.
- операция `machine[regno]`, която проверява дали кола с номер `regno` е регистрирана в паркинга,
- операция за изход <<, която извежда паметта на автомата,
- метод `late(time, hrs)`, който извежда регистрационните номера на всички автомобили, които са прекарали повече от `hrs` часа на паркинга, ако текущото време е `time`. Функцията да връща общата дължима сума за тези коли .

Бонус Класът `ParkingMachine` да се реализира чрез свързано представяне, вместо с разширяващ се масив.

Задача 39. Да се реализира клас `Radar`, който представя паметта на специализиран полицейски радар за засичане на превишена скорост на автомобили. В паметта на радара се пази списък с номерата на преминалите коли (низ от 7 или 8 символа) и за всяка кола се пази засечената скорост в цяло число километри в час. Първоначално радарът може да запази информация за 10 коли, но при препълване на паметта да удвоява капацитета си до достигане на лимита от 1280 коли. За класа да се реализират:

- голямата четворка,
- метод `register(regno, speed)`, който записва информация за новозасечена кола с номер `regno` и скорост `speed`, Ако колата вече е била засичана, запазва се по-високата от двете регистрирани скорости
- метод `hide(regno)`, който изтрива информацията за кола с номер `regno`,
- операция `radar[regno]`, която извежда засечената скорост на кола с номер regno (0, ако такава кола не е засичана),
- операция за изход <<, която извежда паметта на радара,
- метод `speeding(speed)`, който извежда регистрационните номера на всички коли, които са превишили скоростта `speed` и връща средната им скорост.

Класът `Radar` да се реализира чрез свързано представяне, вместо с разширяващ се масив.

Задача 40. Да се реализира шаблон на клас `PriceTag<T>`, който представя етикет с цена на продукт от произволен тип T. За шаблона да се реализират:

- подходящи конструктори, селектори и мутатори;
- операции `>>` и `<<` за въвеждане на етикет от стандартния вход и извеждане на стандартния изход;
- член-функция `discount`, която намалява цената на етикета с даден процент.

Да се реализира шаблон на клас `PriceCatalog<T>`, който описва ценови каталог на продукти от тип T, представен с набор от до 200 етикета с цени на продукти. За шаблона да се реализират:

- подходящ конструктор;
- операция `+=` за добавяне на етикет към каталога;
- операция `<<` за извеждане на информацията в каталога.

Допускаме, че за типа T са реализирани голяма четворка и операции `>>` и `<<`.

Задача 41. Да се реализира абстрактен базов клас `LauncherItem`, който описва елемент на екрана на мобилно устройство и поддържа следните операции:

- `getTitle`, която връща низ, представящ името на елемента;
- `canMove`, която се използва за проверка дали елементът може да бъде местен по екрана.

Да се реализира клас `Launcher`, който представя главния екран на мобилно устройство като правоъгълна матрица от елементи `LauncherItem`. Възможно е всяка дадена позиция в матрицата да е празна, т.е. да няма никакъв елемент на нея. За класа `Launcher` да се реализират следните операции:

- оператор `<<` за извеждане, който извежда на стандартния изход информация за всички елементи на екрана: позиция и име;
- операция `addElement(li, i, j)`, която добавя нов елемент `li` на координати `(i, j)` на екрана. В случай, че тази позиция е заета от друг елемент, операцията добавя елемента на първото свободно място на екрана гледайки отгоре надолу и отляво надясно. Ако на екрана няма свободно място, операцията е неуспешна.
- операция `removeElement(i, j)`, която премахва елемента на координати `(i, j)` от екрана. Ако на тези координати няма елемент или елементът не може да бъде местен, операцията е неуспешна.
- операция `moveElement(i, j, k, l)`, която премества елемента на координати `(i, j)` на нови координати `(k, l)`. Операцията е успешна точно тогава, когато на позиция `(i, j)` има елемент, който може да бъде местен и:
  - на координати `(k, l)` няма елемент; ИЛИ
  - на координати `(k, l)` има друг елемент, който може да бъде местен. В този случай двата елемента разменят местата си;
- операция `clear`, която изчиства екрана.

Да се реализират следните класове, наследници на `LauncherItem`:

- `MobileApp`, който описва стартер на мобилно приложение с дадено име. Елементите от този тип могат да се местят.

- `SystemWidget`, който описва системен елемент. Системните елементи не могат да се местят, и имат едно и също име: “System Widget”.
- `Folder`, който описва папка съдържаща елементи от вид `LauncherItem` в правоъгълна матрица. Папката поддържа всички гореописани операции на клас `Launcher`. Папката може да бъде местена по екрана. Името на папката представлява списък от имената на всички елементи в нея разделени със запетайка.

За гореописаните класове да се реализират подходящи конструктори, селектори и мутатори.

Примерни теми за проекти

Проект 1: Command Line Utilities

Този проект представлява серия от инструменти с които ще се обработват текстови и бинарни файлове. Обработката на текстовите файловете ще включва филтриране на съдържанието по дадени критерии, търсене и сортиране. Ще има включени инструменти за четене и писане във файлове по зададени параметри. Инструментите ще са подходящо направени, така че да позволяват използването на няколко от тях едновременно върху едни данни (файл). Например прочитане на текстов файл и извеждане на всички редове от него, които съдържат определен низ, след което записване на получения резултат в друг файл или извеждане на конзолата.

Задача 1. Напишете клас представящ филтър, който предоставя възможността да се чете вход от конзолата до край на файл ([ctrl] + D/Z). След това може да изведе последователно тези от прочетените редове, в които се съдържа избрана дума. Преценете какви член функции и член данни трябва да има този клас. Демонстрирайте използването му в кратка main функция.

Задача 2. Напишете клас представящ подредена поредица от филтри (FilterChain). Класът ще трябва да може се инициализира с поток за писане и поток за четене. Класът предоставя възможност да се запише филтрираното съдържание от входния поток, преминало последователно през всички филтри, в изходния поток. Добавете подходящи методи за добавяне и премахване на филтрите от класа. Този клас трябва да може да се записва (сериализира) и чете (десериализира) от бинарен файл. Документирайте кода по подходящ начин (коментари + описания).

Задача 3. В това домашно ще трябва да предефинирате изброените оператори за съответния клас, както поне още 4 които не са изброени за класа FilterChain.

За клас Filter, следните оператори:

`operator=  - стандартен`

`operator== - сравнява два филтъра по стринга който филтрират`

`operator!= - обратното на ==`

`operator<< - записва низът за филтриране в поток (низ за филтриране - това което се филтрира от текста)`

`operator>> - чете си низът за филтриране от поток (низ за филтриране - това което се филтрира от текста)`

`operator+= - десен аргумент char добавя аргумента към края на низът за филтриране`

`operator+= - десен аргумент char* добавя дадения низ към края на низът за филтриране`

`operator|  - два аргумента Filter връща FilterChain съставен от аргументите си`

За клас FilterChain, следните оператори:

`operator=  - стандартен`

`operator== - сравнява само филтрите, но не и подредбата (равни FilterChain-ове дават еднакъв резултат при еднакъв вход)`

`operator!= - обратното на == (различни FilterChain-ове дават различен резултат при различен вход)`

`operator+= - добавя Filter към класа`

`operator|  - ляв аргумент FilterChain, десен - Filter - връща FilterChain с добавен десния`

аргумент

`operator+  - два аргумента FilterChain - връща нов, с филтрите от аргументите без повторение`

`operator-= - десен аргумент char*, премахва всички филтри който филтрират подадения низ`

`operator[] - десен аргумент int връща филтър на позиция подаденото число`

`operator[] - десен аргумент char* връща филтър филтриращ подадения низ`

Документирайте и подреждайте кода си.

Задача 4. Сложете следните класове в подходяща йерархия с наследяване:

- WordFilter - досегашният филтър който пропуска редове които съдържат дадена дума
- EncodeFilter - криптира/кодира входен текстов файл по избран от вас начин
- DecodeFilter - декриптира/разкодира входен файл от направен с EncodeFilter
- CapitalizeFilter - променя първата буква на всяка дума в главна от входния текстов файл
- ZeroEscapeFilter
  - работи върху бинарни файлове
  - изхода на филтъра е бинарен
  - в изхода на филтъра не се съдържа byte със стойност 0x00/0
- ZeroUnescapeFilter
  - работи върху бинарни файлове направени от ZeroEscapeFilter
  - изхода е бинарен и е същият какъвто е бил преди да премине през ZeroEscapeFilter

Пояснения:

1. ZeroUnescapeFilter прави обратният процес на ZeroEscapeFilter-а, той трябва да възтанови escpae-натото съдържание.
2. Изхода на EncodeFilter-а може да е текстов или бинарен но DecodeFilter-а трябва да може да възтанови началното съдържание. Начинът по който кодирате или криптирате съдържанието може да е какъвто и да е, стига да е обратим.

Задача 5.

- Реализирайте общ интерфейс за всички филтри.
- Направете FilterChain да поддържа всички филтри които имате до сега.
- Добавете в сегашната йерархия клас SortFilter който трябва да връща редовете от входа сортирани лексикографски
- Демонстрирайте как поне 3 от филтрите работят в един FilterChain обект последователно.

Задача 6.

- Направете поне една демонстрация FilterChain работи с поне един обект от всеки тип филтър.
- Направете поне една демонстрация как FilterChain има поне 5 филтъра и се сериализира и десериализаира правилно.
- Направете по една демонстрация със всяка двойка обратими филтри (Encode/Decode и Escape/Unescape) така че входните данни да минат и през двата филтъра.

Изберете и направете поне единия от следните интерфейси - ако направите и двата ще имате

бонус.

- Направете изпълним файл за всеки от филтрите който работи така че да поддържа pipe-ване

```
sort-filter input-file.txt | line-filter "The D programming language" | encode-filter | zero-escape-filter > output-file.txt
```

- Напревете един общ изпълним файл който да поддържа следния интерфейс

```
CLU input-file.txt output-file.txt --sort --line-filter="The D programming language" --encode --zero-escape
```

Забележка: двата примера за двата интерфейса би трябвало да имат еднакъв изход

Проект 2: Игра с карти

Целта на проекта ви е да реализирате походова игра с карти. Играта се играе от двама играчи, всеки от които притежава предварително зададено тесте. В тестето участват до 10 карти. Всяка карта притежава четири свойства - Атака, Живот, Необходима енергия, Умения (от 1 до 3 на брой). Победител се излъчва когато единия играч остане без карти. За изваждането на карта на "бойното поле" е необходима енергия. Играчите започват с опередена енергия и след всеки ход получават допълнителна такава.

Задача 1. Първата ви задача около този проект е да създадете клас карта. Той трябва да притежава горепосочените свойства като на този етап ще игнорираме уменията. Необходимата енергия се пресмята по формулата E = Ж/100 + А/20 (E - необходима енергия, Ж - живот, А - атака). Класът трябва да притежава конструктор, копиращ конструктор и деструктор. Останалите методи и член-данни трябва да бъдат избрани от вас. Демонстрирайте използването на този клас в кратка main функция.

Задача 2. Имплементирайте нов клас Deck (тесте), които да съдържа масив от карти (произволен брой). За да кажем че тестето е валидно то трябва да съдържа поне 5 карти. Класа Deck трябва да има методи които му позволяват да се записва/инициализира във/от файл (бинарен), ако тестето е валидно. Освен това трябва да поддържате добавяне на нова, изтриване или промяна на съществуваща карта от тестето (респективно от файла). Същото така трябва да имате функция, която да записва 5-те най-добри карти по даден критерии в текстов файл (критерият ще се получава като аргумент на функцията). Ако е нужно допълнете класа карта с нови параметри и функции. Документирайте кода по подходящ начин (коментари + описания).

Задача 3. За класа Card предефинирайте следните оператори:

`operator= - присвояване на карата`

`operator== - проверява дали 2 карти имат еднакви параметри`

`operator!= - проверява дали поне едно поле на 2 карти е различно`

`operator<< - оператор за изход в поток`

`operator>> - оператор за вход от поток`

`operator+= - Приема десен аргумент число и увеличава текущата кръв на карата с него (кръвта не може да надминава максималната)`

`operator-= - Приема десен аргумент число и намалява текущата кръв на карата с него (кръвта не може да е отрицателно число)`

За класа Deck предефинирайте следните оператори:

`operator= - пристояване на дек`

`operator== - проверява дали два дека имат еднакви карти (без значение подредбата)`

`operator!= - проверява дали в два дека има поне 1 различна карата или различен брой карти`

`operator < > <= >= - сравнява сборът от атака и живота на картите в двата дека`

`operator<< - оператор за изход в поток`

`оператор+ - приема карта и дек и връша дек с картите от дека и новата карта(независимо дали`

`отляво седи картата или дека)`

`оператор+ - приема два дека и връща нов дек с картите от двата`

`operator+= - приема десен аргумент карта и я прибавя в дека`

`operator+= - приема десен аргумент дек и прибавя всички карти от него в дека от ляво`

`opertaor[] - приема аргумент цяло число n и връща указател към n-тата карта в дека`

За класа Deck предефинирайте поне още четири оператора по ваше усмотрение.

Задача 4. Съставете подходяща йерархия от наследяване на следните класове:

- BattleField - клас който представя поле което съдържа изиграните карти на двамата играчи. Картите се слагат на полето винаги в най-лявата свободна позиция. На полето всеки играч може да има неограничено количество карти. Ако някоя карта умре трябва всички карти от дясната й страна да се преместят с една позиция на ляво (в масива не трябва да има дупки). Умрялата карта отива в общ масив наречен гробище където се съхраняват всички умрели карти до момента.
- Класа BattleField трябва да може да бъде сериализиран в масив от байтове, който не трябва да съдържа байт избран при конструирането на BattleField-а.
- BattleField-а трябва да може да бъде конструиран чрез сериализирания байт масив като възстанови всички карти и тестета които са били в него преди сериализацията. Десериализацията трябва да става без нужда от знанието на игнорираният байт.
  - Байт (byte) > буква
  - Трябва да може да се сериализират произволно големи полета
- Имплементирайте клас Skill който представлява умение на карта. Всяко умение може да въздейства на една или повече карти (свои или противникови) които се намират на полето или в гробището. Класа Skill трябва да има метод, който приема като аргумент бойното поле, списък с позиците на картите върху който ще въздейства (както и други методи който усигуряват нормалното му функциониране).
- Имплементирайте поне 2 класа, който наследяват Skill И въздействат на собствени карти. Като умението на поне един от класовете действа на повече от 1 карта.
- Имплементирайте поне 2 класа, който наследяват Skill и въздействат на вражеските картиКато умението на поне един от класовете действа на повече от 1 карта.
- Имплементирайте клас, който извиква от гробището, призовават, или забраняват на карта да се завърне от гробището.
- Имплементирайте 2 класа, който се случват с някаква вероятност.
- Имплементирайте 2 класа, който въздействат на поне 3 карти (противникови или собствени, като вие решете какво ще стане ако няма достатъчен брой карти на полето).

Атрибутите и въздействието на всички класове са по ваше усмотрение. Документирайте добре как точно се очаква, да се държат вашите умения, за да не стават недоразумения.

Задача 5.

Създайте два нови типа карта:

- Карта Guard - докато е на полето противниковите карти са задължени да атакуват нея преди всички останали. Изключение правят специалните умения който си действат по нормалния за тях начин.
- Карта Assasin - след като изпълни нормалния си ход нанася 150% щети на героя(или на

карта Guard, ако има такава).

Променете класа Card по подходящ начин така че да е възможно да поддържа до 3 различни умения. Променете класа Deck така че да е възможно да се слагат карти от други типове.

Забележка: Цялата стара функционалност трябва да продължи да работи по начина, по който се очаква(конструктури, оператори, функцианалността от старите домашни).

Задача 6.

- Създайте клас Player който трябва да съдържа задължително Deck , точки живот и точки енергия.
- Допълнете класа BattelField така че да съдържа двама играчи.
- Имплементирайте механиката на битката при следните условия:
- Играчите трябва да имат избор, коя карта да сложат(спрямо това дали имат достатъчно енергия за нея), вие решете как играча получава точките енергия.
- Трябва да имат избор на всеки ход, кое умение на всяка карта да използват.(както и трябва да измислите начин не всички умения да са възможни за ползване на всеки ход).
- Когато карта удари по играча му намаля точките живот.
- Ако играча остане без точки живот умира и другия играч печели принцесата.
- Създайте база(файл) от карти от която всеки играч може да си състави свой Deck.
- Трябва да поддържате Save и Load на играчи(тоест декове и т.н).
- За всичко това създайте подходящ потребителско интерфейс, който да е удобен и лесен за ползване.