Общи изисквания (за задача 1 и 2)

Решенията трябва да отговарят на следните изисквания и ограничения:

- За съхранение на потребителите и предизвикателствата да се използва минимално количество динамична памет. Всички масиви, използвани в класовете, трябва да са с точна дължина.
- Отделните елементи на системата да са представени с класове
- Следвайте добрите ООП практики за реализация на тези класове
- В реализираните класове трябва да са добавени подходящи методи, които са необходими за правилното решение на задачата, въпреки че може да не са явно посочени в условието.
- Решение, което не се компилира или грубо нарушава принципите на ООП, ще бъде оценено с 0 точки.
- За реализацията не е позволено използването на класа `std::string`, както и на контейнерите от STL.

Задача 1: Blackjack

Да се реализира симулатор на популярната казино игра **Блекджек (Blackjack).**
В реализацията задължително трябва да са включени следните класове: карта, тесте и играч. В решението могат да се реализират и други класове, характеристики и операции, освен изброените по-долу.

Картата притежава следните характеристики:

- Боя (спатия, каро, купа, пика)
- Стойност (2, 3, 4, 5, 6, 7, 8, 9, 10, J, Q, K, A)
- Сериен номер (символен низ с максимална дължина 15 символа)

Тестето притежава следните характеристики:

- Последователност от карти
- **Серия** (символен низ с максимална дължина 10 символа). Серийните номера на всички карти в тестето трябва да са уникални и **да започват със серията на тестето**.

Тестето да може да се конструира по следните начини:

- **по подразбиране** – съставя се стандартно тесте с 52 карти (по една от всяка комбинация на боя и стойност), като серията на тестето в този случай е `"Default"`. Картите са подредени в случаен ред.
- **с параметри брой карти `k` (цяло число) и серия `s` (низ до 10 символа)** – съставя тесте от **`k`** на брой случайни карти, като всяка карта се повтаря най-много $\lfloor (k-1)/52 + 1 \rfloor$ пъти (т.е ако имаме до 52 карти, картите не се повтарят, а ако имаме 53 карти, всяка карта може да се среща най-много два пъти). Ако серията `s` не е указана, тя се задава да е равна на низа `"Custom"`.

Над тестето да могат да се изпълняват следните операции:

- **`draw()`** – първата карта се премества на дъното на тестето и се връща като резултат. В частност, ако в тестето има **`k`** на брой карти, след **`k`** пъти извикване на `draw`, то трябва да се върне в първоначалния си вид.
- **`swap(...)`** – по подадени две цели положителни числа се разменят картите с тези поредни номера в тестето. Операцията трябва да се изпълни само ако и двата подадени поредни номера са валидни за тестето.
- **`suit_count(...)`** – по подадена боя се връща броя на картите в тестето с тази боя.
- **`rank_count(...)`** – по подадена стойност се връща броя на картите с тази стойност от все още **неизтеглените** карти.

Играчът притежава следните характеристики:

- **Име –** задължително две имена, състоящи се от латински букви, започващи с главна буква и разделени с точно един интервал;
- **Години –** цяло число в интервала [18, 90] (все пак, това е хазартна игра);
- **Победи –** цяло неотрицателно число, което представя броя на победите на играча;
- **КП (**коефициент на победи) **–** дробно число, в което ще се пази каква част от изиграните си игри е спечелил играчът;
- **ТК –** цяло неотрицателно число, задаващо текущ брой точки за текущо играната игра. При всяко начало на нова игра ТК трябва да има стойност 0.

За целите на задачата ще опишем само част от официалните правила на играта.

Правилата, които трябва да се реализират,са следните:

Всяка карта носи определен брой точки:

- Картите с числова стойност носят толкова точки, колкото е стойността им (от **2** до **10**)
- Картите вале (**J**), дама (**Q**) и поп (**K**) носят по 10 точки
- Асото (**A**) по изключение може да се брои за 1 или 11 точки, като се избира броят точки, който е в полза на играча

Играта се играе с едно тесте от карти и в нея участват играч и раздавач.

Играчът започва с 1 карта (тегли се картата от тестето). Играчът има право да тегли колкото карти иска. Целта му е да събере максимален брой точки, без **да надхвърля 21.** Ако при тегленето играчът надхвърли 21 точки, той губи автоматично и играта приключва. В противен случай, след като играчът приключи с тегленето, е ред на раздавача. Той тегли докато изтеглените от него карти **не надвишат 17 точки**. Ако получените точки са повече от 21, това е загуба за раздавача и победа за играча. Ако обаче те са в интервала [17, 21], то печели този, който е събрал повече точки. При равенство ще считаме, че печели играчът.

Програмата трябва да работи в диалогов режим. При стартирането на програмата трябва да се предложи на потребителя **да създаде нов играч** или да **използва вече съществуващ (като при избора може да се видят статистиките за играча)**. След като избере играч, потребителят трябва да избере дали ще играе със стандартно тесте или със случайно създадено такова със зададен от него брой карти. След това започва играта. Играчът избира на всеки ход дали да тегли или да приключи. След това (**ако играчът не е надхвърлил 21 точки**) се генерира тегленето на раздавача. Програмата трябва да изведе съобщение дали играчът е спечелил или загубил. След това данните за потребителя се актуализират и програмата завърша изпълнението си.
Докато тегли карти, потребителят трябва да има възможност да провери каква е вероятността при следващото теглене на карта точките му да се допълнят точно до 21. При пресмятането на вероятността се абстрахираме от това, че картата на върха на тестето е еднозначно определена и приемаме, че са известни само **неизтеглените** до момента карти в тестето. С други думи, търси се вероятността от все още **неизтеглените** карти да се изтегли карта със стойност (21 - p) точки (p са събраните точки до момента от потребителя). Очевидно, ако 21 - p > 11, то вероятността е 0.
Пример: Ако **p** = 17 и в тестето са останали 25 неизтеглени карти, сред които има три карти на стойност 4, то получената вероятност е 3/25 = 0,12.

Бонус за спец. Информатика, задължителна част за спец. Компютърни науки:
Информацията за всички играчи **трябва да се пази в двоичен файл.**

Пример за изпълнение на програмата (с $ е отбелязан входът на потребителя):

```
Ivan Ivanov 5 0.8
Petar Popov 4 1
Kiril Petkov 2 0.5
Chose a player or enter a new player:
$ Petar Popov
You will play as Petar Popov. Choose the size of the deck:
$ 52
Start!
K(spades) (Points 10)
Hit/Stand/Probability
$ Hit
6(clubs) K(spades) (Points: 16)
Hit/Stand/Probability
$ Probability
0.12
Hit/Stand/Probability
$ Stand
The dealer’s draw: Ace(spades), Jack(spades) (Points: 21)
You lose!
All changes are saved successfully in the file!
```

Задача 2: #STAYHOME

Докато трае самоизолацията и си седим вкъщи (което се оказа много трудно), решихме да се разнообразим с това да се предизвикваме по различен начин в социалните мрежи и под тага #stayhome заваляха и #pushupschallenge, #toiletpaperchallenge, #качисрамнаснимкachallenge и т.н. Програмистите, така ентусиазирани, че и другите сега споделят живота им на самоизолация, решили да зарадват света с ново приложение “StayHomeChallenges”, което ще е само за предизвикателства. И понеже това била идея на екипа по ООП във ФМИ, а идеята е идея за милиони, те решили да поискат помощ от студентите си. Искаме приложението да достигне до всеки, за това трябва да се разработи както и desktop версия, така и сайт, а най-вече и мобилно приложение. Трябва, разбира се, да се поддържа и от всички операционни системи. Работата е много, затова на студентите е възложена най-тежката задача – да напишат цялото ядро на приложението, на база на което ще работят всички негови версии.

Бонус {

Понеже навсякъде има интернет пирати, които копират приложения и ги представят за свои, трябва нашето ядро да има предпазна “система” от това да може да се копира.

}

За да може екипът по ООП да започне да разработва поне beta версии на мобилните приложения, настолното приложение и уеб приложението, към студентите е зададено условието да се поддържат следните основни функционалности:

1. Регистриране на потребители
2. Отправяне на предизвикателства

Все още не знаем дали приложението ще пожъне успех и затова не сме сигурни колко потребители ще се регистрират. Поради това приложението трябва да поддържа произволен брой потребители. Всеки потребител трябва да има име и освен това, за да може всеки по света да има възможност да

използва приложението, не е добре да се налага ограничение колко дълго да е то. При регистрация потребителят трябва да въведе основна информация: години (до 90) и мейл адрес– до 100 символа. За да може потребителите с еднакви имена, години и имейл адреси да бъдат различавани, системата трябва сама да слага уникален идентификационен номер при успешна регистрация.

Докато екипът по ООП пише за различните приложения форми за регистрация с полета и бутони, ядрото на системата ще работи с текстови команди. Регистрацията ще става със следната команда:

```
registration Integralcho 19 integral@fmi-sofia.bg
```

Потребителят трябва да може да се регистрира без да се налага да предоставя част от личната си информация като години и имейл адрес:

```
registration Integralcho 19
```

```
registration Integralcho integral@fmi-sofia.bg
```

Разбира се, за да може успешно да бъде предизвикан даден потребител, той трябва задължително да въведе името си.

Следващата функционалност, която трябва да поддържа системата, е отправяне на предизвикателства. Един потребител предизвиква останалите по следния начин:

`challenge` *<името на потребителя, който предизвиква> <таг на предизвикателство> <имената на предизвиканите, отделени с интервал>*

Следните проверки трябва да се извършат преди предизвикателството да се регистрира успешно в системата:

- Всеки от потребителите (предизвикващ и предизвикани) трябва да е вече регистриран в системата;
- Тагът задължително започва със знака `#.` Той идентифицира уникално предизвикателството и има максимална дължина от 31 символа;

- Ако предизвикателството не е било отправяно досега, то се добавя в системата и неговият статус става `"new"`. Ако е било отправено преди, то не се регистрира наново, но се увеличава броя на отправянията му и се сменя статусът му, ако е нужно. Ако е било отправяно между 2 и 10 пъти, статусът му трябва да бъде `"quite recently"`, а от 11 нагоре – `"old"`.

За да знаем дали едно предизвикателство е интересно и успяващо, то има и рейтинг (дробно число), който варира в интервала [-5.0, 10.0]. Екипът по ООП предизвиква студентите (поради дефицит на памет) предизвикателството да се представя с не повече от 40 байта в паметта.

Всеки потребител пази списък от предизвикателствата, които са отправени към него и които все още не е изпълнил. Той изпълнява предизвикателствата със следната команда:

`finish` *<таг на предизвикателство> <идентификационен номер на потребителя> <с каква оценка го оценява>*

За да разберем какъв е идентификационния номер на потребителя, неговият профил може да се прегледа със следната команда:

`profile_info` *<име на потребител>*

Като резултат се извежда цялата въведена при регистрация информация за потребителя. Ако някое от полетата не е въведено при регистрация, за негова стойност се изписва `"Unknown"`.

След като предизвикателството е изпълнено, потребителят вече не пази информация за него, а рейтингът му се обновява. Рейтингът се пресмята като средно аритметично от всички дадени оценки. Понеже все още системата е в тестов режим, броят на отправяния на предизвикателство и броят на изпълнения няма да надвишава 65535.

Разбира се, приложението трябва да има и стена, на която може да се виждат всички предизвикателства и да се подреждат по определен начин. Това става със следната команда:

`list_by` *<как да са подредени>*

като имаме следните възможности за подредба: `newest` (по ред на отправяне, първо най-скорошните), `oldest` (по ред на отправяне, първо най-старите), `most_popular` (по брой отправяния, първо най-често отправяните).

Бонус за спец. Информатика, задължителна част за спец. Компютърни науки:

За да може да изтестваме по-бързо системата, може да се поддържат и следните команди:

```
load users.txt
```

и

```
load challenges.bin
```

които зареждат в системата съответно потребители (от текстов файл) и предизвикателства (от двоичен файл). Командата `load` трябва автоматично по разширението да определя дали зарежда потребители или предизвикателства.

Пример за изпълнение на програмата (с $ е отбелязан входът на потребителя):

$ registration Integralcho 19 integral@fmi-sofia.bg

Registration Successful. Please check your email.

$ registration Mariika 25 qkataMara95@abv.bg

Registration Successful. Please check your email.

$ registration DqdoToshko 92 rakiq@gmail.bg

Registration Fail. You are too old for this App.

$ registration Milcho 37

Registration Successful.

// Следващите примери ще са с потребители, които можем да смятаме, че са вече регистрирани или са прочетени от файла users.txt

$ challenge Integralcho #toiletpaper Ivan Dragan Marrika

Integralcho successfully challenged!

$ challenge Milcho izqshlutqchushka Mariika Integralcho

Sorry! Invalid tag of the challenge.

$ challenge Hristo #reshiSudokuto DqdoToshko

Sorry, some of the users may not be registered!

// Следващите примери ще си представим, че са отправени предизвикателства, които са регистрирани успешно и са отправяни повече от веднъж.

$ finish #toiletpaper 2 8.6

Well done! May the challenge be with you!

$ finish #toiletpaper 8 -0.6

Well done! May the challenge be with you!

$ finish #pushupschallenge 13 7.6

The given id isn't valid. The user with this id do not exist.

$ finish #pushupschallenge 9 17.6

Sorry. Invalid rating! The challenge is not completed, yet.

$ profile_info Integralcho

Searching for users…

1) Name: Integralcho

   Age: 19

   Email: integral@fmi-sofia.bg

   Id: 1

2) Name: Integralcho

   Age: 37

   Email: Unknown

   Id: 10

3) Name: Integralcho

   Age: Unknown

   Email: Unknown

Id: 14

$ profile_info BabaStanka

Searching for users…

Nothing come out! User with this name do not exist!

$ list_by most_popular

Tag	Rating	Status	Total	Done
#gledaiSerial	10.0	old	1024	1024
#napishiDomashnoPoOOP	-3.5	old	311	258
#HodiBezMaskaBezDaTeGlobqt	4.2	`quite recently`	8	0
#kachiSnimkaKatoBebe	5.5	new	1	0

Задача 3: Hearthstone or any other TCG

Trading card game (TCG) е игра, в която двама играчи конструират по едно тесте от карти. По време на играта двамата играчи се редуват да теглят карти от тестетата си и да ги играят като всяка карта представлява някакво действие или събитие. Такива игри са Hearthstone, Yu-gi-oh, Magic: the gathering и т. н. Конкретно в Hearthstone всеки играч започва с определен брой точки живот. Всеки ход протича по следния начин: играчът тегли карта от тестето си, след това играе карти от ръката си. Картите в Hearthstone са 2 типа: Minion и Spell. Minion е карта, която остава на терена след играенето си – играчът ги ползва за да понижи живота на опонента си и да предпази собствения си живот. Spell е карта, която има някакъв еднократен ефект и след изпълнението му изчезва от терена. Всяка карта си има цена (mana), което ограничава броя карти, които могат да се играят всеки ход. Печели играчът, който пръв понижи живото на опонента си до 0. Помислете каква йерархия от карти може да направите. По какъв начин можете да представите играчите, логиката на ходовете и терена? Как ще се осъществяват взаимодействията между картите, терена и играчите? Каква структура ще ползвате за представяне на тестето на играчите.

Задача 4: Fire emblem or any other tactical RPG

Tactical role-playing game е игра, в която двама играчи местят фигури върху разграфен терен. Теренът можете да си го представите като таблица n на m (подобно на шахматна дъска). Всеки играч поставя определен брой фигури в някоя зона на терена. След това играчите се редуват да местят своите фигури и с тях да атакуват фигурите на противника си, спазвайки правилата за движение и атака на съответната фигура.

Конкретно във Fire emblem всяка фигура представлява герой със различни характеристики – име, точки живот, атака, брой полета които може да извърви на ход и т.н. Също има различни видове герои със специфични свойства. Например Fighter може да атакува само съседите си, Archer може да атакува само герои на разстояние 2, Cavalier може да се движи повече от другите герои и т.н. Има и някои полета от терена със специални свойства като стени, през които не може да се мине и гори, които забавят героите. Печели играчът, който победи всички герои на опонента си. Алтернативен вариант е всеки да си има фигура-цар, и играч да печели, ако победи фигурата цар на противника си.

Помислете каква йерархия ще ползвате за да представите различните герои. Как ще се осъществяват движенията и битките? Какви класове ще са нужни за представяне на играчите и логиката на ходовете им? Как ще се реализира терена и различните препятствия по него, ако има такива?