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

Теми за проекти

курс Обектно-ориентирано програмиране
за специалност Компютърни науки
Летен семестър 2017/2018 г.

Обща информация за проектите

Проектът е цялостна задача, която трябва да решите с помощта на познанията по C++, получени през летния семестър. Правилата за завършване на курса “ООП практикум” с избрания от вас проект са следните:

1. Срок за работа по проекта - 3 дни преди датата за защитата на проектите по ООП практикум.
2. На изпита трябва да предадете:
    - Документация на проекта;
    - Изходен код на решението;
    - Изпълним файл с решението;
    - Няколко примера, подбрани от Вас, които демонстрират работата на задачата.
3. В решението на задачата задължително трябва да използвате парадигмата на ООП. Решението, в което кодът е процедурен, има лоша ООП архитектура и т.н. се оценява с нула точки.
4. Документацията на проекта трябва да съдържа:
    - Кратък анализ на задачата и Вашия подход за решение (на какви стъпки сте разделили решението, какъв метод или алгоритъм сте избрали, как сте се справили с конкретен проблем).
    - Кратко описание на класовете, създадени от Вас за решение на задачата - описание на член-данните и член-функциите им и начина на използване на класовете.
    - Идеи за бъдещи подобрения
5. По време на защитата трябва да разкажете в рамките на 10 минути Вашето решение и да демонстрирате работата на програмата с подготвени от Вас данни.

6. По време на защитата се очаква да можете да отговорите на различни въпроси, като например: (1) каква архитектура сте избрали, (2) защо сте избрали именно нея, (3) дали сте обмислили други варианти и ако да — кои, (4) как точно работят различните части от вашия код и какво се случва на по-ниско ниво и др.
7. Възможно е да ви дадем малка задача за допълнение или промяна на функционалността на проекта ви, която вие трябва да реализирате на място за максимум 1 час.
8. Невъзможност да реализирате малката задача на място означава, че не познавате добре проекта си и сте ползвали чужда помощ за реализацията му. Последното ще се отрази негативно на оценката ви.
9. Критерии за оценка на проекта (% от оценката):
    - Правилно решение на задачата - 10 точки
    - Добре подбрана структура на класовете решаващи задачата - 20 точки
    - Подреден и читаем код – 10 точки
    - Подходяща и ясна документация – 10 точки
    - Представяне на проекта – 10 точки
    - Възможност за бърза реализация на малка промяна в проекта (познаване на кода) и отговаряне на въпроси – при невъзможност, оценката става 0 точки.
10. По желание може да се направи презентация за реализирания проект.

Проект 1
Diablo 0.5
(изготвил: Александър Шумаков)

Една от най-популярните RPG поредици се казва Diablo. Третото попълнение бе дългоочаквано и придоби голям успех след пускането на първото разширение: Reaper of Souls. Но зад успеха на тази вече култова класика стои дълга история на разработка – първата част е пусната на пазара на 31.12.1996г. По онова време компютърните игри не са били с толкова полирани визуални и звукови качества и ефекти, както са днес. Били са прости – базирали са се на една основна идея и са се развивали около нея. Някои от игрите, смятани за най-велики и въздействащи и до днес, всъщност не представляват нищо повече от един конзолен интерфейс.
Вашата задача е да направите примитивна версия на играта Diablo.
За целта ще ви е нужно да реализирате следните класове и интерфейси:

Character
Character (интерфейс): този интерфейс ще представлява основните данни и функционалности за всеки един герой;
Данни:

- **Име (name):** названието на един бъдещ легендарен войн;
- **Ниво (level):** кое ниво е героят ни в момента;
- **Жизнени точки (HP):** оставащият живот в героя;
- **Сила (strength):** колко физическа сила притежава;
- **Интелект (intelligence):** магическата способност на даденият герой;

Функции:

- **Нападение (attack):** нанасяне на щета върху враг;
- **Отбрана (defend):** получаване на щети, като от точките за атака на противника се изкарват толкова проценти, колкото е **strength** показателя на героя;
- **Порастване (levelUp)**: функция, която се извиква автоматично при убиване на **2^<level>** противника; подсилва показателите спрямо пропорционалното им разпределение при създаването на героя. За тази цел се разпределят общо 5т. на 2 показателя. (Забележка: жизнените точки се покачват автоматично с 10% от текущата им стойност)
  Например:
  При начална статистика от сорта на **10 strength** и **4 intelligence**, то ще имаме следното пресмятане:
  <нова стойност на показател> ::= [(<първоначална стойност>/14) * 5] + <текуща стойност>

Героите могат да бъдат следните няколко вида:

- **Варварин (Barbarian):** физически могъщ и свиреп, майстор на близкия бой и контактните оръжия. Притежава специална енергия, наречена **ярост (rage)**. Яростта се генерира когато варваринът удари противник (+2т.) или когато противникът уцели варваринът (+3т.). Капацитетът на герой от този вид е до 100т. ярост.
  Начални стойности:
  100HP, 10 strength, 3 intelligence, 0 rage
  Варваринът се бие по следния начин: изразходва всичката ярост, която притежава, в началото на всяка битка, като яростта му дава бонус (ярост / 5) процента атака за цялата битка с даденият противник. Базовата щета, която нанася варваринът, е <точки сила> + 0.2 * <точки интелект>.
  Натрупаните по време на конкретна битка точки ярост се използват за следващата битка.
- **Магьосник (Sorcerer):** мъдър, спокоен и непоклатим психически, почитател на древните тъмни учения. Притежава специална енергия, наречена мана (**mana**). Маната се възвръща на максимум след всяка битка. Максималният капацитет мана, който един магьосник може да има, се увеличава с 10% при всяко порастване (подобно на жизнените точки).
  Начални стойности:
  70HP, 4 strength, 12 intelligence, 100/100 (100 точки в началото от 100 точки начален капацитет) mana
  Магьосникът се бие по следния начин: при всяка атака нанася щети, определени от следната формула: [50% + (<текуща мана> / <капацитет мана>) * ¾] * <точки интелект>. След нападение, точките мана на разположение на магьосника спадат с 10% от капацитета.
- **Ловец на глави (bounty hunter):** изобретателен и ловък, експлоатиращ и най-незабележимите пукнатини в защитата на врага. Не притежава специална енергия, но пък разполага с ловкост (**agility**). Точките ловкост се повишават с 4 на всяко порастване.
  Начални стойности:
  80HP, 9 strength, 6 intelligence, 10 agility
  Ловецът на глави се бие по следния начин: всяка трета атака, която нанася в дадена битка, нанася на врага <точки ловкост> + 0.4 * <точки интелект> + 0.6 * <точки сила> точки щета. Останалите 2 атаки нанасят 0.8 * <точки ловкост> точки щета.

Enemy

Enemy (интерфейс): този интерфейс ще предоставя основните данни и функционалности за всеки един противник;

Данни:

- **Име (name):** названието на един свиреп звяр
- **Жизнени точки (HP):** оставащият живот във врага
- **Сила (strength):** колко физическа сила притежава
- **Интелект (intelligence):** магическата способност на даденото чудовище

Функции:

- **Нападение (attack):** нанасяне на щета върху герой
- **Отбрана (defend):** получаване на щети, като от точките за атака на противника се изкарват толкова проценти, колкото е strength показателя на чудовището.

Враговете могат да бъдат следните няколко вида:

- **Скелет (Skeleton):** малоумно чудовище, задвижвано от незнайна сила; не представлява заплаха за който и да е истински герой
  Начални стойности:
  32HP, 3 strength, 0 intelligence
  Скелетът се бие по следния начин: нанася щети колкото е силата му + 1/10 от интелекта на врага
- **Еретик (Heretic):** изгубена душа в тялото на човек, подмамен от пътя на черната магия; залъгва своите опоненти и ги напада емоционално
  Начални стойности:
  20HP, 1 strength, 4 intelligence
  Еретикът се бие по следния начин: нанася щети колкото са точките на най-ниският показател на опонента му + точките интелект на самият еретик.
- **Макромант (Necromancer):** полужив труп със задълбочени познания в забранените изкуства, играещ си с живота и смъртта; може да призовава поданик скелет, който да го отбранява, докато той се готви да нанася удари
  Начални стойности:
  65HP, 2 strength, 7 intelligence
  Макромантът се бие по следният начин: на първият рунд от битката (а след това и на всеки три рунда) той призовава нов свой поданик скелет, който да поема ударите вместо него. Нанася щети колкото са точките HP на скелета-поданик / 10 (ако има такъв, ако няма – 0) + своите точки интелект.
- **Демон (Diablo):** първичното изчадие, породило всичко зло на планетата; породено от ерес и смърт (т.е. да се имплементира като наследник на класовете **Еретик** и **Скелет**)
  Начални стойности:
  350HP, 8 strength, 4 intelligence

Демонът се бие по следният начин: редува по една атака като на скелет и като на еретик.

Да се имплементира интерактивен режим на играта, който включва:

- Избор на герой
- Карта (двумерен масив), по която са разположени разнообразни противници;
    - когато се опитаме да стъпим върху поле на противник, то героят го атакува и след това, ако противникът е все още сред живите, той отвръща с атака;
    - ако се опитаме да стъпим на невалидно поле - да се връща пояснително съобщение защо това е “невалиден ход” и да се дава възможност за повторно въвеждане.
- Стандартен изход на актуална информация за картата и последиците от последния ход на играча

Проект 2
ШАХ

(изготвил: Александър Шумаков)

Задачата ви е да напишете програма, която позволява да се провежда игра на шах между двама играчи.
За целта, създайте следните класове и интерфейси със съответните функционалности:

Фигура

Фигура (Figure): абстрактен клас, който ще се разширява от различните видове фигури в играта:

- **Пешка (Pawn)**: може да се придвижва само едно поле напред (две ако досега не се е движила); може да атакува фигури, които са на 1 квадратче по диагонал от нея
- **Топ (Rook)**: може да се придвижва хоризонтално или вертикално до достъпните квадратчета и да атакува фигури, намиращи се по тях.
- **Кон (Knight)**: може да се придвижва Г-образно по дъската и да атакува фигури, намиращи се на квадратчетата, на които може да стъпи.
- **Офицер (Bishop)**: може да се придвижва по диагонал и да атакува фигури на съответните позиции.
- **Царица (Queen)**: може да се придвижва като Топ или като Офицер; да се реализира чрез наследяване на тези 2 класа;
- **Цар (King)**: може да се придвижва и атакува само фигури само в непосредствено съседство;

Всички фигури да притежават информация за собственика им и информация за противниковите фигури, които са взели по време на играта (коя фигура, на коя позиция, в кой ход).
Да се реализират следните функции:

- **printStats** - извежда информация за всички взети фигури
- **move -** приема като параметри координатите, на които искаме да преместим фигурата, и връща **true**, ако фигурата може да се премести на конкретна позиция, и **false** в противен случай.

Да се реализират нужните за тези имплементации помощни функции и да се използват помощни променливи по собствена преценка.

Дъска

Дъска (Board): клас, който ще държи информацията за разположението на фигурите по дъската в даден момент. Да се имплементира с нужната функционалност по собствена преценка.

Игра

Игра (Game): клас, който ще представлява клиентски интерфейс за играта.
Интерфейсът трябва поддържа следните команди, въведени от клавиатурата:

- **move x1 y1 x2 y2** – при възможност мести фигурата от позиция (x1, y1) на позиция (x2, y2);
- **print** - извежда на екрана игровата дъска
- **stats x1 y1** – при възможност извежда информация за фигурата на позиция (x1, y1);
- **undo** – връща играта в състоянието преди последния ход.

При невъзможност да се изпълни някоя от функционалностите, да се извежда интуитивно съобщение, което пояснява на потребителя защо не може да се изпълни дадената операция и да му позволи да въведе нова команда.
Потребителите трябва да могат да въвеждат команди редувайки се и командите трябва да отговарят по функционалност на съответният потребител, например “move” не може да се изпълни на чужда фигура.

Проект 3
Програма за управление на персонала на фирма

(изготвил: Александър Шумаков, Данаил Димитров, Петър Събев)

Да се напише програма, която да може да поддържа данните за служителите на софтуерна компания.

Всеки служител на компанията се дефинира по следния начин:

- Лична данни - Пълно име, Адрес по лична карта и ЕГН
- Дата на постъпване на работа
- Пряк ръководител
- Длъжност

Компанията разполага с 3 длъжности:

- Анализатор
- Програмист
- Лийд

Всяка длъжност има 3 степени:

- Ниво 1
- Ниво 2
- Ниво 3

За всеки програмист трябва да се запише:

- Името на проекта, по който работи в момента (името може да се променя)

За всеки Анализатор трябва да имаме информация за:

- Името на проекта, по който работи
- Имейлите на клиентите, с които комуникира

Лийдът изпълнява едновременно ролята на другите две длъжности

Програмата трябва да разполага с меню, което позволява:

- Въвеждане на информация за фирмата
- Извеждане на тази информация
- Добавяне на служител
- Извеждане на данните за служител
- Да може да се уволни служител
- Да се изведе информация за всички служители, които заемат дадена длъжност
- Промяна на данни за служител

Приема се, че във фирмата постъпват организационни промени. Целият персонал на фирмата се разделя на екипи. Всеки служител може да участва точно в един екип. Възможно е да има екипи, в които не участват хора.
Всеки екип трябва да дефинира по следния начин:

- Ръководител
- Името на проекта, по който се работи
- Кои служители участват в него - реализацията зависи вас

Към менюто се добавят следните опции:

- Създаване на нов екип
- Прехвърляне на служител от един екип към друг
- Изтриване на екип
- Показване на данни за всички екипи ()

Да се предефинират операторите за вход и изход на екип.
Не е възможно да има хора, които да не са назначени в нито един екип.

Проект 4
БДЖ

(изготвил: Александър Шумаков)

Да се напише програма, която да поддържа административната система на БДЖ. За целта са ни нужни следните абстракции:

- **Гара (Station) –** притежава име (символен низ с произволна дължина), набор от влакове, които в момента са на гарата (масив от указатели от тип Train с произволна дължина), и приоритет (цяло число между 1 и 10).
- **Влак (Train) –** притежава вагони със съответен капацитет (динамичен масив от цели числа, където елементът с индекс i отговаря на това колко места има във вагон с номер i), скорост (цяло число, км/ч) и маршрут (опашка от тип Гара).

Да се предефинират операторите за вход и изход за влаковете.

Да се създадат следните видове влакове:

- **Бърз влак (QuickTrain)** – влак, който има коефициент (дробно число между 1 и 3), което показва колко по-бързо от обикновен влак се движи**.**
- **Директен влак (DirectTrain)** – влак, който притежава собствен приоритет (подобно на гара) и спира само на гарите от своя маршрут с приоритет не по-нисък от собствения (ако няма такива гари, спира само на последната спирка)
- **Експресен влак (Express) –** представлява бърз, директен влак

БДЖ Линия (RailwayLine) се дефинира по следния начин:

- Информация от коя до коя гара се движи даден влак
- В колко часа потегля влака
- В колко часа пристига влака

БДЖ Разписание (RailwaySchedule) – съдържа информация за всички БДЖ Линии**.**

Програма трябва да съдържа меню, което позволява да :

- **Да се добавят и премахват станции/влакове.**

При добавяне на влак да се въвеждат:

- Час на потегляне от първоначалната му спирка и да се създават съответните БДЖ Линии по информацията, която имаме за влака - скорост, начална спирка, дестинации (т.е. да се съставят всички възможни от-до комбинации).

Да се реализира функционалност, която симулира движението на влакове до даден час (премества ги на правилната гара).

Да се реализира функционалност, която позволява да видим валидните линии (от въведен час нататък).

Проект 5
Библиотека Matrix

(изготвил: Петър Събев и Данаил Димитров)

Да се създаде библиотека, която реализира основни действия за работа с матрици - поне 4 смислени математически действия по Ваше желание.
Основни класове, които библиотеката трябва да предоставя са:

- **Matrix** – клас, който реализира матрица с произволна размерност;
- **DenseMatrix** – клас, който реализира плътна матрица;

Библиотеката поддържа две реализации на плътна матрица:

- **DenseMatrix1D** – плътната матрица (DenseMatrix), която съхранява данните си в едномерен масив.
- **DenseMatrix2D** – плътната матрица (DenseMatrix), която съхранява данните си в двумерен масив.

Библиотеката трябва да предоставя следните възможности, в допълнение на избраните математически действия:

- Създаване на матрица.
  Създаването трябва да бъде възможно по:
  - брой редове и колони
  - брой редове, колони и масив (едномерен или двумерен)
- Достъп до елемент на матрицата посредством зададени индекси за номер на ред и номер на колона;
- Промяна на елемент на матрицата посредством зададени индекси за номер на ред и номер на колона;

В допълнение на това трябва да напишете програма, представяща смислена употреба на библиотеката. А каква по хубаво от програма за шифроване, прилагащи.
В нашия свят изпълнен с непослушни уши, искаме да предаваме съобщения като не желаем другите да ги разбират (освен получателят им). За целта ще използваме различни шифроващи подходи.

А какво представлява един шифроващ подход и какво може да прави?

- Приема съобщение (във вид на низ) и го криптира като го преобразува до различен (криптиран) низ.
- Приема криптирано съобщение във вид на низ и го преобразува до първоначалният низ.

За реализацията на тази задача ще се възползваме от средствата на линейната алгебра в лицето на матрици.
Дизайнът на приложението трябва да е такъв, че да позволява при нужда

лесно да можете да добавите поддръжка на нови криптиращи подходи.
Един пример за такъв алгоритъм можете да намерите тук

Проект 6
NFS (the game)

(изготвил: Данаил Димитров)

Тъй като много харесваме високите скорости, особено когато сме удобно настанени вкъщи, ще си направим нещо, което да се справи с този глад... може би.
Ще си направим много опростена версия на игра за състезания с коли. Това ще бъде повече симулатор, отколкото игра, но е достатъчно близко. В нашата „игра“ ще имаме коли, пари... състезания и състезатели.

Като за начало, нашата игра ще е компактно прибрана в един клас.
В основата на всичко са **състезателите (играчи ли шофьори)**. Част от тях ще бъдат управляване автоматично, докато друга част ще бъдат управлявани от нас (можем да се съберем няколко приятели да играем на една машина).

В началото се генерират n на брой играчи, като първите m са тези, управлявани от нас, като **n и m са избират при стартиране на играта**.
От тези състезатели:

- Всеки започва с без пари - нали вече си има кола. Ще трябва да си ги печели
- Всички състезатели започват с еднакво умение.
- Всеки състезател притежава поне една **кола** и започва със случайна такава (случайни но сходни по мощност)**.**

Всеки **състезател притежава:**

- Тип
  Типът може да бъде:
  - Човек
    - На всеки рунд се изисква от него да избере действие
  - Компютър
    - Имат по-висок коефициент на случайност
    - Трупат умение по-бързо
- Умение - число, което подобрява шансовете за победа на състезател. Печели се чрез участие в състезания (опит).
- Пари
  - Използват се за закупуване на коли и части
  - Печелят се от победи в състезания
  - Състезател от тип компютър, няма нужда от пари, защото не могат да купуват
- Кола

    - Използва се за участие в състезания
    - Може да бъде закупена различна кола от магазинът за коли

Една **кола** има:

- Тип
  Типът може да бъде:
    - Нормална
    - Nitro кола - колата получава бонус точки на всяко 3-то състезание (това е времето за презареждането на Nitro-то в колата)
- Мощност - число

Магазинът за коли притежава набор от коли за продажба. За улеснение този списък е предварително дефиниран в програмата.
Магазинът може да:

- Показва списъка от коли, от който купувачът да си избере, като в това представяне трябва да е достатъчно интуитивно - да се вижда цена и основни характеристики.
- Да продава кола, която състезателят си е избрал.

Защо му е кола на състезател, ако няма да я кара? Всеки състезател може да участва в състезания.

Как се извършва състезание?

- За сега ще имаме само един вид състезания, а именно - един срещу един (1vs1).
- Играч избира срещу кого да се състезава и всичко започва! За да можем да си изберем противник ще трябва и да виждаме съществуващите играчи. Би било добре да показваме повече информация за тях от самото име.
- Слез установяване на противниците, победителят се определя много просто - **сравнявайки техните коефициенти**. Коефициентът на играч се определя по формула и използва следните елементи: умение на състезател, качества на колата и шанс (защото винаги може да духне вятър точно когато не трябва).
  Пример за такава*:*
    - *нека състезател 1 притежава*
        - ***x = умение на играчът***
        - ***y = мощност на автомобилът му***
        - ***z =*** *генерира се случайно число* ***в интервал 0 - 20****;*

    - *то коефициентът му ще бъде - **(x+y) * (100 + z)%***
      Тоест, късметът може да повиши коефициента на играч до 20%!
- Останалите състезатели, на случаен принцип играят по между си всеки път когато ние играем.

Състезателите управлявани автоматично, нямат нужда от магазин... всъщност те просто участват в състезания и трупат умение.
Играем докато ни харесва!

А как ще работи играта?

- Генерират се състезателите
- Генерират се колите като:
    - Искаме те да са достатъчен брой с нарастваща мощност
    - Искаме да има и от двата типа (с и без nitro)
- На всеки рунд ние можем да правим следните:
    - Участваш в състезание, избирайки противник. За целта ще трябва да показваме съществуващите играчи по удобен начин. Това ти носи пари и умение
    - Да си закупуваш по-добра кола за твоята
    - Останалите играчи ги управлява „компютърът“ и се бият по между си на случаен принцип

Проект 7
Issues tracking system (Kanban)
(изготвил: Данаил Димитров)

Тъй като нашият екип от програмисти расте все повече, разработката на приложения се затруднява все повече. Хората забравят кой по коя задача работи, забравят какво има да се прави, а ръководителят на проекта се затруднява в отговора на въпроса дали проектът се движи по план и колко време още ще се разработва.
С цел да подобрим работата на екипите, ще създадем система, която да се грижи за част от нещата.
Нашата система за "Следене на задачите" в различните проекти ще се грижи за следните проблеми - да имаме списък от всички за задачи за различните проекти (както ненаправени, така и приключени); ще показва прогресът по тези задачи и ще ги визуализира по подходящ начин;

В основата на нашето приложение са самите задачи, които трябва да бъдат решавани, защото решаването на един голям проблем изисква разбиването му на подзадачи.

Една **задача** има:

- Описание;
- Изпълнител (кой решава задачата в момента);
- Състояние - възможните състояния се определят от дъската, в която се намира задачата
- Проект или компонент от кода, към който се отнася задачата (просто string)
- Блокираща задача, ако такава съществува (това е задача, която трябва да бъде направена преди текущата);
- Тип
    - **Бъг** - описва проблем в съществуващата имплементация
    - **Story** - задача представена през очите на клиентът. Това са задачите за създаване на нова функционалност
    - **Epic** - това е задача съставена от множество задачи

След като разполагаме със списък от всички задачи (или така наречения backlog), време е да ги представим по удобен начин за работа на отделни екипи по тях.
За целта ще използваме представяне във вид на разграфена „дъска“. Пример за такава - board. Всяка колона показва състоянието на задачите в нея.

Задачите се движат последователно между колоните, като започват от най-лявата. **Ще поддържаме множество от такива дъски (за различните екипи).**

Една **дъска** се дефинира чрез:

- Име
- Тип
  Типът може да бъде:
  - **Simple** - съдържа три графи, както следва: “ToDo”, “In progress”, “Done”
  - **Reviewable** - съдържа допълнително поле за review
  - **Testable** - съдържа допълнителна колона: “For Testing”
  - **Complex** - съдържа пет графи, обединявайки “Testable” и “Reviewable”

Всяка дъска е празна при създаването си и черпи задачите си от backlog-а
Винаги първата и последната колона на всяка от дъските са фиксирани на “ToDo” и “Done”

Какво можем да правим?

- За задачите
  - Създаване на задачи, които се настаняват удобно в backlog-а.
    - при създаването им се дава възможност да бъдат зададени всички от изброените свойства на задачите, но само част са задължителни
  - Да виждаме всички задачи
- За дъските:
  - Да създаваме дъска - създава се чрез списък от задачи
  - Да виждаме всички дъски
  - Да показваме задачите в дъска
    - Задачите могат да бъдат показани лесно по редове (вместо колони)
    - Всеки от типовете задача си има символ, за по-лесно ориентиране - !, *, #
  - Да променяме изпълнителят на задачата
  - Да променяме състоянието на задача (да придвижваме задача напред по състоянията)

Как изглежда workflow за използване на програмата:

- Първо се създават задачи, по удобен начин

- Създава се дъска
- В дъската се добавят задачи, от тези в backlog-а
- За някоя задача се избира изпълнител и се задава
- Когато изпълнителят започне да работи по нея, я мести в “In progress”
- И така нататък...

Целим основно покриване на функционалностите, а не безкрайни детайли във всяко. Разбира се можете да допълвате на воля.

Проект 8
Имоти

(изготвил: Данаил Димитров и екип)

Да се дефинира абстрактен клас имот **Estate**.
Всеки се дефинира чрез:

- Адрес на имота;
- Собственик;
- Цена;
- Площ;

Да се реализира функционалност, която извежда информацията за даден имот в конзолата.
Всеки имот може да бъде един от следните типове:

- Апартамент (**Flat**) - съдържа информация за брой стаи и етаж.
- Къща (**House**)- съдържа информация за брой стаи, брой етажи, площ на двора.
- Парцел (**Plot)** - съдържа информация за налични комуникации (списък от елементи от изброим тип Communication (вода, ток, телефон, близост до път, канализация) и площ.

За да съществува един имот той трябва да бъде регистриран.
Да се дефинира клас, който поддържа списък на регистрираните недвижими имоти **RealEstates**, съхраняващ контейнер от указатели към обекти клас **Estate**, в който ще се съхраняват обекти от тип имот.

Ние можем, както да регистрираме, така и да премахваме имот от списъка с регистрирани имоти.

За да можем да намерим определен имот се налага да използваме брокери.
Всеки такъв разполага със списъка от регистрираните имоти, но използва различно оценяване на имот при търсене.
Всички брокери **завишават цените на имотите, за да включат техния хонорар.**
Всеки брокер се дефинира чрез:

- **Име**
- **Процент**
- **Тип**
  Различаваме три типа брокера:

- **HelpfulBroker / ThoughfulBroker - притежава списък с ВИП обяви**, които се визуализират преди останалите обяви и подобрява шансът даден имот да бъде представен.
- **DishonestBroker -** притежава списък с ВИП обяви, **но ги представя с подобрени цена и площ на случаен принцип**.
- **SimpleBroker** - не се държи по-различно от обикновена търсачка.

Да се реализира функционалност, която позволява добавяне и премахване на имот към списъка с ВИП имоти в брокерите от първите два типа.

За да преглеждаме имоти, ще ни се наложи да си изберем брокер, който да ни обслужи. Добре, а как търсим?

След като си изберем брокер, можем да подадем нашите критерии (задаването критерии, може да става чрез меню или по друг подходящ и избран от вас начин) за търсене и той да ни представи списък от предложения.
За тази цел програмата трябва да поддържа следните функционалности:

- Търсене по различни изисквания на клиента - вид на имота, цена от зададен интервал, площ, налични комуникации, дадено населено място.
- Извеждане на всички имоти от зададен вид, сортирани по цена в низходящ ред.
- Извеждането на имотите трябва да става в табличен вид.

Да се демонстрира работата на класа **RealEstates**, като се дефинира обект от този клас и списък от брокери и се предостави възможност за различни справки, които демонстрират методите за търсене.

Проект 9
Media player

(изготвил: Данаил Димитров)

Налага ни се да си създадем нещо, с което да си пускаме различни видове музикални файлове за моментите, когато имаме нужда (когато ни е скучно). Тогава нека си направим нещо за справяне с такива!

За съжаление трудно ще можем да постигнем всичко със сегашните си знания, затова ще направим малко по-особена версия на такъв.

А какво ще има нашият „играч на медия“?

Музикален файл, който си има:

- Име
- Изпълнител
- Продължителност
- **audioData** - съдържа бинарната информация (в реалния свят) за музикални файлове. Засега ще я пазим като стринг, представяйки си, че може да е нещо по-сложно, което ще реализираме по-късно.
- Тип - **MP3 или flac**

Съществуват също и втори тип музикални файлове - видео музикален файл.
Видео музикален файл
Един такъв притежава допълнителни качества:

- Освен **audioData**, съхранява информация и за картината - **videoData -** под същата форма (стринг)
- По-различни типове - **MP4, AVI**

Ние можем да си добавяме свободно такива файлове в нашият плейър, който послушно ще си ги запазва.

А как ще ги пускаме? - за целта ще се възползваме от така наречените „декодери“, чиято идея е да преведат бинарната информацията от файлът до нещо „чуваемо“ (или „видяемо“). За съжаление ние няма как да го чуем затова **ще импровизираме чрез проста визуализация**:

- За **звуков файл:**
    - **flac** - принтира **audioData** отпред назад
    - **mp3** - принтиране на **audioData** отпред назад, но с прескачане на всеки 3-ти символ (малко орязано)
- За **видео файл**:

    - **MP4** - визуализира звука (**audioData**), като **mp3**, а **videoData** по същият начин, но след звука
    - **AVI** - визуализира звука (**audioData**) като **flac**, а **videoData -** преди звука

Бихме желали да можем лесно да добавяме нови формати файлове и съответните „декодери“ за тях.

Добре, вече разполагаме с много медия. Време е да добавим възможност за разделяне, а именно плейлисти.
Един плейлист има:

- Име
- Множество от медия файлове
- Възможности за **добавяне/премахване на файлове**
- Възможности да бъдат пуснати последователно записите в него. Тук се появяват два подхода:
    - **Normal play** - файловете се изпълняват последователно;
    - **Shuffled play** - файловете се изпълняват в разбъркан вид;

В нашият плейър ние можем:

- Да добавяме / премахваме медия;
- Да прегледаме всички налични медии;
- Да потърсим медия по име или име на изпълнител;
- Да пуснем медия;
- Да създаваме плейлист;
- Да добавяме/премахваме елементи в плейлист;
- Да изслушаме конкретен плейлист;

Нашият плейър ще бъде конзолен като ще предоставя следните команди:

- play <*media id*>
- add
    - Изискваме да бъдат въведени данните за медията.
    - Ако типа на медията предполага, че е видео, ще искаме да се въведе и **videoData.**
- remove <*media id*>
- show all
- show by artist <*musician name*>
- show by name <*song name*>
- create playlist
    - Изисква име на плейлиста.
- playlist add <*playlist id*> <*media id1*> <*media id2*> ...

    - Можем да добавяме много файлове наведнъж.
- playlist remove <*playlist id*> <*media id1*> …

Проект 10
SUSI

(изготвил: Александър Шумаков)

Задачата ви е да напишете програма, която симулира информационната система **СУСИ**. За целта е нужно да се реализират следните класове и интерфейси:

Курс

Курсът (Course) трябва да съдържа **име на предмета**, за който се е водил курсът, **година** на провеждане, **множество от записани потребители** (не повече от 200) и съответстващите им **оценки** (възможно да не са цели числа), както и **ръководител** на курса; да се имплементира функционалност за разглеждане на оценки в курса, за промяна или поставяне на оценка на конкретен записан потребител на системата, за добавяне на потребител (при неизчерпана квота); да се предефинира оператора за изход (<<), така че да може да извежда информация за даден курс (ръководител, година на провеждане, участници);

Потребител

Потребител (User) представлява интерфейс, който ще бъде реализиран от роли:

- **Студент (Student)** – трябва да притежава **факултетен номер**, **настоящ курс** и **име на специалността**, в която е; също така, трябва да може да вижда оценката си в даден курс, ако е записан;
- **Преподавател (Teacher)** – трябва да има възможност да вижда оценките на курсовете, които води, както и да добавя оценки към тези курсове;
- **Асистент (Tutor)** - асистентът е студент, който преподава; той трябва да има достъп до функционалността за студент в курсовете, в които е записан като студент, и до функционалността за преподавател в курсовете, в които е преподавател.

Всеки потребител има потребителско име (**username**), парола (**password**) и фактическо име (**name**). Паролата не трябва да се пази в оригиналния си формат, а трябва да се пази **hash**-ът ѝ според някаква фиксирана **hash функция** (по собствена преценка; за повече информация: тук).

Да се дефинира член-функция на интерфейса **printGrades**, която по име на курс принтира оценката на дадения потребител (ако е студент и е вписан в курса) или оценките на всички потребители (ако е преподавател).

Да се дефинира член-функция **changeGrade**, която по подадени име на

потребител, име на курс и нова оценка променя текущите данни в курса (ако потребителят, извикал функцията е преподавател) или извежда съобщение за грешка (ако е обикновен потребител).

Интерфейс

Потребителски интерфейс (UI) – връзката между потребител и информационната система; трябва да съдържа множество от потребители, множество от курсове и да поддържа следните команди:

- **login <username> <password>** - опитва да промени текущо логнатия потребител на този с конкретно зададените потребителско име и парола;
- **view-grades <course_name>** - извиква съответната функция за текущия потребител;
- **change-grade <user_name> <course_name> <new_grade>** - опитва се да промени оценката на даден потребител;
  Да се извежда подходящо съобщение за резултата от функцията. Да се имплементират и използват помощни функции по собствена преценка.

Проект 11
Динамичен масив и итератори

(изготвил: Петър Събев)

Целта на проекта е да бъдат реализирани шаблонен клас - динамичен масив - Vector, подобен на std::vector и итератор от STL.

Класът Vector трябва да бъде шаблонен и да бъде реализиран в отделен header файл. В допълнение на това трябва да е изпълнено правилото на петте. Да се използва **operator new (или placement new – по желание)** и съответстващите им за освобождаване на паметта.

По отношение именуването на методите **се спазва интерфейсът на векторът (std::vector) от стандартната библиотека и дефинираното там описание**. Вашият клас трябва да съдържа следните методи:

- **Конструктори, деструктори и оператори за присвояване**;
- **operator[]** - връща псевдоним към елемент по подаден индекс:
- **operator[]** - връща константен псевдоним към елемент по подаден индекс;
- **push_back** - добавя елемент в края на динамичния масив;
- **pop_back** - изтрива елемент в края на динамичния масив;
- **size()** - връща броя елементите;
- **resize(int newSize)** - променя размера на динаичния масив, така че да стане newSize;
- **empty()** - проверява дали има елементи;
- **back()** - връща стойността на елемента в края на Vector;
- **clear()** - изтрива цялото съдържание на Vector;

Наред с горе изброените методи, трябва да бъдат реализирани и още **поне 3 метода по Ваш избор**.

Итератор представлява описание на типове, които могат да бъдат използвани за обхождане на елементите на даден контейнер. Вашата задача е да реализирате поне два вида итератори: единият **RandomAccessIterator**, другият по Ваш избор. Наред с това е необходимо да приложите подходящи тип итератор в реализацията на динамични масив и да предоставите интерфейс за достъп. Изисква се итераторите да бъдат направени по подобие на тези в STL и подходящо приложени.

Да се покаже примерна употреба с написването на смислена програма, създаваща инстанции на вашия динамичен масив с голям брой елементи (поне 1000 елемента) и използваща поне веднъж поне половината от методите на динамичния масив.

По желание (бонус):

- *Да се използва **placement new**;*
- *Да се подсигури правилно поведение при изключителни ситуации чрез сигнализиране и обработване на изключения;*
- *Опитайте се да минимизирате изискванията към шаблонните типове.*