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

Теми за проекти

курс Обектно-ориентирано програмиране
за специалности Информатика, Информационни системи и Компютърни науки
летен семестър 2019/2020 г.

Обща информация за проектите

Проектите се оценяват по редица от критерии, част от които са описани по-долу.
Тъй като курсът се фокусира върху обектно-ориентираното програмиране и неговата реализация в езика C++, най-важното изискване за проектите е те да са изградени съгласно добрите принципи на ООП. Решението, в което кодът е процедурен, има лоша ООП архитектура и т.н. се оценява с нула точки. Всички проекти, които работят с дати, да ги представят във формата ISO 8601. Други важни критерии за оценка на проектите са:

- Дали решението работи коректно, съгласно спецификацията. Решение, което не работи и/или не се компилира носи минимален брой (или нула) точки.
- Дали решението отговаря на заданието на проекта.
- Каква част от необходимата функционалност е била реализирана.
- Дали решението е изградено съгласно добрите практики на обектно-ориентирания стил. Тъй като курсът се фокусира върху ООП, решения, които не са обектно-ориентирани се оценяват с нула или минимален брой точки.
- Оформление на решението. Проверява се дали кодът е добре оформен, дали е спазена конвенция за именуване на променливите, дали е добре коментиран и т.н.
- Дали решението е било добре тествано. Проверява се какви тестове са били проведени върху приложението, за да се провери дали то работи коректно. Очаква се по време на защитата да можете да посочите как сте тествали приложението, за да проверите дали то работи коректно и как се държи в различни ситуации.

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.

Оценката на всеки от проектите се формира от онази негова част, която е била самостоятелно разработена от вас. Допустимо е да използвате код написан от някой друг (напр. готова библиотека или помощ от ваш приятел/колега), но (1) той не носи точки към проекта и (2) това трябва да бъде ясно обявено както при предаването, така и при защитата на проекта, като ясно обозначите коя част от проекта сте разработили самостоятелно. Това означава, че:

1. Използваният наготово код трябва да се маркира ясно, като поставите коментари на подходящи места в кода си.
2. По време на защитата трябва да посочите кои части сте разработили самостоятелно и кои са взети от други източници.

Както е написано по-горе, когато в проекта си използвате чужд код, сам по себе си той не ви носи точки. Допълнителни точки могат да се дадат или отнемат, според (1) способността ви за внедряване на кода във вашето решение (напр. в случаите, когато се използва външна библиотека) и за това (2) дали добре разбирате какво прави той.

Съдържание

Работа с командния ред **5**

Проект 1: Приложение за работа с електронни таблици **8**

Проект 2: Работа със SVG файлове **13**

Проект 3: XML Parser **16**

Проект 4: Недетерминиран краен автомат **19**

Проект 5: Контекстно-свободна граматика **21**

Проект 6: Бази от данни **23**

Проект 7: Traveller’s app **26**

Проект 8: Джурасик парк **28**

Проект 9: Големи числа **30**

Проект 10: Библиотека **31**

Проект 11: Хотел **34**

Проект 12: Склад **36**

Проект 13: СУСИ **38**

Проект 14: Билети **41**

Проект 15: Личен календар **43**

Проект 16: JSON парсер **45**

Проект 17: Растерна графика **47**

Проект 18: Dungeons & Dragons **51**

Проект 19: Star Wars Universe 0.1 **54**

Проект 20: Шах **57**

Бонус проекти: игри **60**

Snake 60

Alien Attack 60

Sokoban 61

Tetris 61

Breakout 61
Xonix 61
Pacman 62
Mine Sweeper 62
Pong 62
Lander 62
Arcade Volleyball 63
Frogger 63
The Game of Life 63
2048 64
Занимателна математика 65

Работа с командния ред

(отнася се за проекти с номера 1–6 и 10–17 & 19)

Вашата програма трябва да позволява на потребителя да отваря файлове (open), да извършва върху тях някакви операции, след което да записва промените обратно в същия файл (save) или в друг, който потребителят посочи (save as). Трябва да има и опция за затваряне на файла, без записване на промените (close). За целта, когато програмата ви се стартира, тя трябва да позволява на потребителя да въвежда команди и след това да ги изпълнява.

Когато отворите даден файл, неговото съдържание трябва да се зареди в паметта, след което файлът се затваря. Всички промени, които потребителят направи след това трябва да се пазят в паметта, но не трябва да се записват обратно, освен ако потребителят изрично не укаже това.

Във всеки от проектите има посочен конкретен файлов формат, с който приложението ви трябва да работи. Това означава, че:

1. то трябва да може да чете произволен валиден файл от въпросния формат;
2. когато записва данните, то трябва да създава валидни файлове във въпросния формат.

Както казахме по-горе, потребителят трябва да може да въвежда команди, чрез които да посочва какво трябва да се направи. Командите могат да имат нула, един или повече параметри, които се изреждат един след друг, разделени с интервали.

Освен ако не е казано друго, всяка от командите извежда съобщение, от което да е ясно дали е успяла и какво е било направено.

Дадените по-долу команди трябва да се поддържат от всеки от проектите. Под всяка от тях е даден пример за нейната работа:

Open

Зарежда съдържанието на даден файл. Ако такъв не съществува се създава нов с празно съдържание.

Всички останали команди могат да се изпълняват само ако има успешно зареден файл.

След като файлът бъде отворен и се прочете, той се затваря и приложението ви вече не трябва да работи с него, освен ако потребителят не поиска да запише обратно

направените промени (вижте командата save по-долу), в който случай файлът трябва да се отвори наново. За целта трябва да изберете подходящо представяне на информацията от файла.

Ако при зареждането на данните, приложението ви открие грешка, то трябва да изведе подходящо съобщение за грешка и да прекрати своето изпълнение.

```
> open C:\Temp\file.xml
Successfully opened file.xml
```

Close

Затваря текущо отворения документ. Затварянето изчиства текущо заредената информация и след това програмата не може да изпълнява други команди, освен отваряне на файл (Open).

```
> close
Successfully closed file.xml
```

Save

Записва направените промени обратно в същия файл, от който са били прочетени данните.

```
> save
Successfully saved file.xml
```

Save As

Записва направените промени във файл, като позволява на потребителя да укаже неговия път.

```
> saveas "C:\Temp\another file.xml"
Successfully saved another file.xml
```

Help

Извежда кратка информация за поддържаните от програмата команди.

```
> help
The following commands are supported:
open <file> opens <file>
close             closes currently opened file
save              saves the currently open file
saveas <file>     saves the currently open file in <file>
help              prints this information
```

```
exit              exists the program
```

Exit

Излиза от програмата

```
> exit
Exiting the program...
```

Проект 1: Приложение за работа с електронни таблици

Представяне на данните

Данните на една таблица ще записваме в текстов файл по следния начин:

1. Всеки ред във файла представя отделен ред в таблицата.
2. Всеки ред във файла съдържа данни разделени със запетаи. Тези данни се интерпретират като стойностите в клетките на реда.
3. Всеки ред в таблицата може да съдържа различен брой клетки. Затова и всеки ред във файла може да съдържа различен брой елементи разделени със запетаи.
4. Празен ред във файла представя празен ред в таблицата. (т.е. ред, в който всички клетки са празни).
5. Между две запетаи във файла може да няма никакви данни. По този начин се представя празна клетка.
6. Между данните и запетаите може да има произволен брой празни символи (whitespace).

Така за една таблица може да има различни представяния. Например таблицата:

10	20	30	40
10		1000	
	10		

може да се представи по следните начини (възможни са и други представяния):

```
10, 20, 30, 40

10,,1000,
,,,
,10
```

```
10, 20 , 30  , 40

10,   , 1000,
  ,    ,    ,
  , 10
```

Типове данни в таблицата

Всяка клетка в таблицата има тип, като в една таблица може да има едновременно клетки от различни типове. Вашето приложение трябва да може да поддържа следните типове:

Цяло число – поредица от цифри, без никакви други символи между тях. В началото на числото може да има знак '+' или '-'. Например:

```
123
-123
+123
```

Дробно число – поредица от цифри, следвана от символ за точка и след нея друга поредица от цифри. В началото на числото може да има знак '+' или '-'I. Например:

```
123.456
-123.456
+123.456
```

Символен низ (стринг) – поредица от произволни символи оградени в кавички. Подобно на низовете в C++, ако искате да включите символа за кавичка в даден низ, трябва да го представите като \", а ако искате да включите наклонена черта, трябва да я представите като \\. Например:

```
"Hello world!"
"C:\\temp\\"
"\"This is a quotation\""
```

Формула – формулата винаги започва със символ за равенство. В нея могат да участват следните операции: събиране (+), изваждане (-), умножение (*), деление (/) и степенуване (^). Във формулата могат да участват или числа или препратки към клетки в таблицата. Ако във формулата участва препратка към клетка, на това място в изчислението трябва да се използва стойността съхранена в дадената клетка. Повече информация за формулите е дадена по-долу.

Нужна функционалност

След като приложението отвори даден файл, то трябва да може да извършва посочените по-долу операции, в допълнение на общите операции (`open, close, save, save as`, `help` и `exit`):

`print`	Извежда съдържанието на таблицата на екрана
`edit`	Редактира съдържанието на дадена клетка. За целта потребителят въвежда текст, който ще бъде новото съдържание на клетката. Забележете, че по този начин може да се промени типът на дадена клетка, например от число, тя може да стане формула.

Както беше казано в общата за всички проекти информация, ако при зареждането на данните, приложението ви открие грешка, то трябва да изведе подходящо съобщение за грешка и да прекрати своето изпълнение. Съобщението трябва да подскаже на потребителя какво не е наред във входните данни. Например:

- Ако липсва запетая трябва да се изведе на кой ред и след кой символ липсва запетаята;
- Ако съдържанието на дадена клетка е от неизвестен тип, трябва да се изведе на кой ред и коя колона е клетката и какво точно е некоректното съдържание. Например нека предположим, че на ред 2, колона 5, потребителят е въвел 123.123.123. Приложението ви може да изведе например следното съобщение: *“Error: row 2, col 5, 123.123.123 is unknown data type”*.

Извеждане на таблицата на екрана

При извеждане на заредената таблица (командата print), данните в колоните трябва да се подравнят. Между отделните колони трябва да се поставят символи за отвесна черта (|). По-долу е даден пример за входен файл и възможно негово извеждане:

Входен файл	Извеждане

`10, "Hello world!", 123.56`
`"\"Quoted\""`
`1, 2, 3, 4`
	`10 | Hello world! | 123.56 | |`
	`"Quoted" | | | |`
	`1 | 2 | 3 | 4 |`

Редактиране на клетки

Командата Edit трябва да позволява (с подходящи параметри) на потребителя да променя стойностите на отделните клетки. Това става като се укажат реда и колоната на клетката, която искаме да променим, а също и каква стойност да запише в нея. Потребителят може да въведе произволен тип данни, който се поддържа от вашата програма (например цяло число, дробно число, низ, формула и т.н.).

Ако потребителят въведе неправилни данни, приложението ви не трябва да променя нищо в таблицата, а само да изведе на екрана съобщение, че са въведени неправилни данни. В този случай приложението ви НЕ трябва да прекратява своето изпълнение.

Формули

Номерата на редовете и клетките в таблицата започват от 1. Препратка към ред <N> и колона <M> в таблицата се записва така: R<N>C<M>. Например клетката в ред 10 и колона 5 се представя като R10C5.

В дадена формула могат да участват единствено:

1. Литерали: цели или дробни числа.
2. Препратки към произволни типове клетки.

При сметките важат следните правила:

1. Ако в дадена формула участват само числа, то сметката се извършва по традиционните правила на аритметиката. Като специален случай можем да отделим делението на две цели числа. В такъв случай не бива да губите остатъка и резултатът трябва да бъде дробно число (например 1 делено на 2 дава резултат 0,5).
2. Ако в дадена формула участва низ, той трябва да се конвертира до число. Това става по следния начин: Ако низът съдържа само цифри или поредица от цифри, символ точка и друга поредица от цифри, той се конвертира до съответното число. Всички други низове се конвертират до нула. Например:

Низ	Конвертирана стойност
“123”	123
“123.456.789”	0
“123.456”	123.456
“Hello world”	0
“123abc”	0

3. Ако в дадена формула участва празна клетка, тя се конвертира до нула. Това важи и за клетки, чиито координати надхвърлят размерите на таблицата.
4. Ако в дадена формула има грешка (например деление на нула), приложението ви не трябва да прекъсва своето изпълнение. Вместо това, когато то извежда таблицата на екрана, в съответната клетка се извежда ERROR, вместо получен резултат.

По-долу е дадена примерна таблица. В нея клетките в жълт цвят са от тип число. Клетките в зелено са от тип символен низ:

	Колона 1	**Колона 2**	**Колона 3**

Ред 1	10	Hello world!	123.56
Ред 2	123		

По-долу са дадени формули, които се оценяват в примерната таблица по-горе. За всяка формула е дадена и нейната оценка:

Формула в клетката	**Реално извършена сметка**	**Стойност на клетката**	**Коментар**
= 10 + 10	10 + 10	20	
= R1C1 + R1C3	10 + 123.56	133.56	
= R1C1 * R1C2	10 * 0	0	Низът „Hello world!“ се конвертира до нула
= R1C1 * R2C1	10 * 123	1230	Низът „123“ се конвертира до 123 Клетката на ред 2, колона 2 е празна В таблицата няма ред 200 и колона 200. Считаме, че тя е празна. т „123“ се конвертира до 123
= R1C1 * R2C2	10 * 0	0	Клетката на ред 2, колона 2 е празна
= R1C1 * R200C1	10 * 0	0	В таблицата няма ред 200 и колона 200. Считаме, че тя е празна.
= 10 / 0	10 / 0	ERROR	
= 10 / R1C2	10 / 0	ERROR	
= R1C1 / R1C2	10 / 0	ERROR	

Проект 2: Работа със SVG файлове

В рамките на този проект трябва да се разработи приложение, което работи със файлове във [Scalable Vector Graphics (SVG) формат](). Приложението трябва да може да зарежда

фигури от файла, да извършва върху тях дадени операции, след което да може да записва промените обратно на диска.

За улеснение, в рамките на проекта ще работим само с основните фигури (basic shapes) в SVG. Приложението ви трябва да поддържа поне три от тях. Например можете да изберете да се поддържат линия, кръг и правоъгълник. За повече информация за това кои са базовите фигури, вижте https://www.w3.org/TR/SVG/shapes.html.

Също така, за улеснение считаме, че координатната система, в която работим е тази по подразбиране: положителната полуос X сочи надясно, а положителната полуос Y сочи надолу.

Дизайнът на приложението трябва да е такъв, че да позволява при нужда лесно да можете да добавите поддръжка на нови фигури.

Когато зареждате съдържанието на един SVG файл, трябва да прочетете само фигурите, които приложението ви поддържа и можете да игнорирате всички останали SVG елементи.

След като заредите фигурите, потребителят трябва да може да изпълнява дадените в следващия раздел команди, които добавят, изтриват или променят фигурите.

Когато записвате фигурите във файл, трябва да генерирате валиден SVG файл

Операции

След като приложението отвори даден файл, то трябва да може да извършва посочените по-долу операции, в допълнение на общите операции (`open, close, save, save as`, `help` и `exit`):

`print`	Извежда на екрана всички фигури.
`create`	Създава нова фигура.
`erase <n>`	Изтрива фигура с пореден номер `<n>`.
`translate [<n>]`	Транслира фигурата с пореден номер `<n>` или всички фигури, ако `<n>` не е указано.
`within <option> ...`	Извежда на екрана всички фигури, които изцяло се съдържат в даден регион. Потребителят може да укаже чрез `<option>` какъв да бъде регионът – кръг (`circle`) или правоъгълник (`rectangle`)

Примерен SVG файл figures.svg

```
<?xml version="1.0" standalone="no"?>
```

```
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"
  "http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
<svg>
  <rect x="5" y="5" width="10" height="10" fill="green" />
  <circle cx="5" cy="5" r="10" fill="blue" />
  <rect x="100" y="60" width="10" height="10" fill="red" />
</svg>
```

Пример за работа на програмата

```
> open figures.svg
Successfully opened figures.svg

> print
1. rectangle 5 5 10 10 green
2. circle 5 5 10 blue
3. rectangle 100  60 10 10 red

> create rectangle -1000 -1000 10 20 yellow
Successfully created rectangle (4)

> print
1. rectangle 1 1 10 20 green
2. circle 5 5 10 blue
3. rectangle 100 60 10 10 red
4. rectangle 1000 1000 10 20 yellow

> within rectangle 0 0 30 30
1. rectangle 5 5 10 10 green
2. circle 5 5 10 blue

> within circle 0 0 5
No figures are located within circle 0 0 5

> erase 2
Erased a circle (2)

> erase 100
There is no figure number 100!

> print
1. rectangle 5 5 10 10 green
2. rectangle 100 60 10 10 red
3. rectangle 1000 1000 10 20 yellow
```

```
> translate vertical=10 horizontal=100
Translated all figures

> print
1. rectangle 105 15 10 10 green
2. rectangle 200 70 10 10 red
3. rectangle 1100 1010 10 20 yellow

> save
Successfully saved the changes to figures.svg

> exit
Exit
```

Проект 3: XML Parser

Да се напише програма, реализираща четене и операции с XML файлове.
Характеристиките на XML елементите, поддържани от програмата, да се ограничат до:

- идентификатор на елемента
- списък от атрибути и стойности
- списък от вложени елементи или текст

Да се поддържат уникални идентификатори на всички елементи по следния начин:

- Ако елементът има поле “id” във входния файл и стойността му е уникална за всички елементи от файла, да се ползва тази стойност.
- Ако елементът има поле “id” във входния файл, но стойността му не е уникална за всички елементи от файла, да се ползва тази стойност, но към нея да се конкатенира някакъв низ, който да допълни идентификатора до уникален низ. (например, ако два елемента имат поле id=”1”, то единият да получи id=”1_1”, а другият - id=”1_2”)
- Ако елементът няма поле “id” във входния файл, да му се присъедини уникален идентификатор, генериран от програмата.

След като приложението отвори даден файл, то трябва да може да извършва посочените по-долу операции, в допълнение на общите операции (`open, close, save, save as`, `help` и `exit`):

`print`	Извежда на екрана прочетената информация от XML файла (в рамките на посочените по-горе ограничения за поддържаната информация). Печатането да е XML коректно и да е “красиво”, т.е. да е форматирано визуално по подходящ начин (например, подчинените елементи да са по-навътре)
`select <id> <key>`	Извежда стойност на атрибут по даден идентификатор на елемента и ключ на атрибута
`set <id> <key> <value>`	Присвояване на стойност на атрибут
`children <id>`	Списък с атрибути на вложените елементи

`child <id> <n>`	Достъп до n-тия наследник на елемент
`text <id>`	Достъп до текста на елемент
`delete <id> <key>`	Изтриване на атрибут на елемент по ключ
`newchild <id>`	Добавяне на НОВ наследник на елемент. Новият елемент няма никакви атрибути, освен идентификатор
`xpath <id> <XPath>`	операции за изпълнение на прости XPath 2.0 заявки към даден елемент, която връща списък от XML елементи

Минимални изисквания за поддържаните XPath заявки

Примерите по-долу са върху следния прост XML низ:

```
<people>
	<person id=”0”>
		<name>John Smith</name>
		<address>USA</address>
	</person>
	<person id=”1”>
		<name>Ivan Petrov</name>
		<address>Bulgaria</address>
	</person>
</people>
```

- да поддържат оператора / (например “person/address” дава списък с всички адреси във файла)
- да поддържат оператора [] (например “person/address[0]” два адресът на първия елемент във файла)
- да поддържат оператора @ (например “person(@id)” дава списък с id на всички елементи във файла)
- Оператори за сравнение = (например “person(address=”USA”)/name” дава списък с имената на всички елементи, чиито адреси са “USA”)

Забележка: За проекта не е позволено използването на готови библиотеки за работа с XML. Целта на проекта е да се упражни работата със структурирани текстови файлове, а не толкова със самия XML. **Внимание:** Не се изисква осигуряване на всички условия в XML и XPath спецификациите! Достатъчно е файловете да “приличат на XML” (както файла в горния пример, който не е валиден XML), а завките да “приличат” на XPath.

Бонуси:

- да се реализират XML namespaces
- да се реализират различните XPath оси (ancestor, child, parent, descendant,...)

Проект 4: Недетерминиран краен автомат

(само за спец. Компютърни науки)

Да се реализира програма, която поддържа операции с недетерминиран краен автомат с Ɛ-преходи. над азбука, състояща се от цифрите и малките латински букви.

Автоматите да се сериализират по разработен от Вас формат. Всеки прочетен автомат да получава уникален идентификатор.

След като приложението отвори даден файл, то трябва да може да извършва посочените по-долу операции, в допълнение на общите операции (`open, close, save, save as`, `help` и `exit`):

`list`	Списък с идентификаторите на всички прочетени автомати
`print <id>`	Извежда информация за всички преходи в автомата
`save <id> <filename>`	Записва автомат във файл
`empty <id>`	Проверява дали езикът на автомата е празен
`deterministic <id>`	Проверява дали автомат е детерминиран
`recognize <id> <word>`	Проверява дали дадена дума е в езика на автомата
`union <id1> <id2>`	Намира обединението на два автомата и създава нов автомат. Отпечатва идентификатора на новия автомат
`concat <id1> <id2>`	Намира конкатенацията на два автомата и създава нов автомат. Отпечатва идентификатора на новия автомат
`un <id>`	Намира позитивна обвивка на автомат и създава нов автомат. Отпечатва идентификатора на новия автомат
`reg <regex>`	Създава нов автомат по даден регулярен израз (теорема на Клини). Отпечатва идентификатора на новия автомат

- да се реализира мутатор, който детерминира даден автомат

- да се реализира операция, която проверяват дали езикът на даден автомат е краен

Проект 5: Контекстно-свободна граматика

(само за спец. Компютърни науки)

Да се реализира програма, която поддържа операции с контекстно-свободна граматика, използваща главни латински букви за променливи (нетерминали) и малки латински букви и цифри за терминали и с недетерминирани стекови автомати.

Граматиките да се сериализират по разработен от Вас формат. Всяка прочетена граматика да получава уникален идентификатор.

След като приложението отвори даден файл, то трябва да може да извършва посочените по-долу операции, в допълнение на общите операции (`open, close, save, save as`, `help` и `exit`):

`list`	Списък с идентификаторите на всички прочетени граматики
`print <id>`	Извежда граматиката в подходящ формат. За всяко правило да се отпечата пореден номер
`save <id> <filename>`	Записва граматиката във файл
`addRule <id> <rule>`	Добавя правила
`removeRule <id> <n>`	Премахване на правило по пореден номер
`union <id1> <id2>`	Намира обединението на две граматики и създава нова граматика. Отпечатва идентификатора на новата граматика
`concat <id1> <id2>`	Намира конкатенацията на две граматики и създава нова граматика. Отпечатва идентификатора на новата граматика
`chomsky <id>`	Проверява дали дадена граматика е в нормална форма на Чомски
`cyk <id>`	Проверява дали дадена дума е в езика на дадена граматика (CYK алгоритъм)
`iter <id>`	Намира резултат от изпълнението на операцията “итерация” (звезда на Клини) над граматика и създава нова граматика. Отпечатва идентификатора на новата граматика
`empty <id>`	Проверява дали езикът на дадена контекстно-свободна граматика е празен

`chomskify <id>`	Преобразува граматика в нормална форма на Чомски. Отпечатва идентификатора на новата граматика

Проект 6: Бази от данни

Да се реализира програма, поддържаща операции с прости бази от данни. Базите данни се състоят от серии от таблици, като всяка таблица е записана в собствен файл. Базата данни е записана в главен файл (каталог), които съдържа списък от таблиците в базата данни, като за всяка таблица е зададено име и файл, в който таблицата е записана.

Поддържани типове данни

Всяка “колона” на таблица в базата данни има тип, като в една таблица може да има едновременно колони от различни типове. Вашето приложение трябва да може да поддържа следните типове:

Цяло число – поредица от цифри, без никакви други символи между тях. В началото на числото може да има знак '+' или '-'. Например:

```
123
-123
+123
```

Дробно число – поредица от цифри, следвана от символ за точка и след нея друга поредица от цифри. В началото на числото може да има знак '+' или '-'. Например:

```
123.456
-123.456
+123.456
```

Символен низ (стринг) – поредица от произволни символи оградени в кавички. Подобно на низовете в C++, ако искате да включите символа за кавичка в даден низ, трябва да го представите като \", а ако искате да включите наклонена черта, трябва да я представите като \\. Например:

```
"Hello world!"
"C:\\temp\\"
"\"This is a quotation\""
```

Освен конкретна стойност, дадена клетка в даден ред на таблицата може да е “празна”. Такива клетки да се обозначават специално и да е изписват като “NULL”.

След като приложението отвори даден файл, то трябва да може да извършва посочените по-долу операции, в допълнение на общите операции (`close`, `save`, `save as`, `help` и `exit`):

`import <file name>`	Добавя в базата данни нова таблица от файл. Във файла е записана информация за типа на всяка колона. **Всяка таблица има име**. При опит за зареждане на таблица с име, което съвпада с името на някоя вече заредена таблица, системата да дава грешка. Добавената таблица се записва в каталога на базата от данни.
`showtables`	Показва списък с имената на всички заредени таблици
`describe <name>`	Показва информация за типовете на колоните на дадена таблица
`print <name>`	Показва всички редове от дадена таблица. Да се реализира диалогов режим, позволяващ съдържанието на таблицата да се преглежда по страници (такива, че се събират на един екран) със следните команди: следваща страница, предишна страница, изход.
`export <name> <file name>`	Записва таблица във файл
`select <column-n> <value> <table name>`	Извежда всички редове от таблицата, които съдържат стойността “value” в клетката с дадения пореден номер. Да се реализира извеждане по страници
`addcolumn <table name> <column name> <column type>`	Добавя нова колона (с най-голям номер) в дадена таблица. За всички съществуващи редове от таблицата, стойността на тази колона да е празна.
`update <table name> <search column n> <search value> <target column n> <target value>`	За всички редове в таблицата, чиято колона с пореден номер <search column n> съдържа стойността <search column value> се променят така, че колоната им с пореден номер <target column n> да получи стойност <target value>. Да се поддържа стойност NULL.
`delete <table name> <search column n> <search value>`	Изтрива всички редове в таблицата, чиято колона <search column n> съдържа стойността <search column value>
`insert <table name> <column 1> ⋯ <column n>`	Вмъква нов ред в таблицата със съответните стойности
`innerjoin <table 1> <column`	Извършва операцията Inner Join над две таблици

n1> <table 2> <column n2>	спрямо колоните <column n1> в първата таблица и <column n2> във втората. Създава нова таблица и извежда идентификатора и.
rename <old name> <new name>	Преименува таблица. Отпечатва грешка, ако новото име не е уникално.
count <table name> <search column n> <search value>	Намира броя на редовете в таблицата, чиито колони съдържат дадената стойност
aggregate <table name> <search column n> <search value> <target column n> <operation>	Извършва дадена операция върху стойностите от колоната <target column n> на всички редове, чиито колони с номер <search column n> съдържат стойността <search value>. Възможните операции са sum, product, maximum, minimum. Системата да дава грешка, ако колоните не са числови.

Проект 7: Traveller’s app

Автор: Стела Маринова

Пътуването е по-приятно, ако е споделено с приятели. Напишете приложение, което да позволява на любителите пътешественици да споделят снимки, изживявания и впечатления.

Нека при отваряне на приложението да има две възможни опции:

- **регистрация на нов потребител** - въвежда се потребителско име, парола и имейл адрес. Създава се запис в базата данни с потребители. Създава се нова лична база данни на потребителя.
- **вписване във вече съществуващ профил** - след правилно въведени име и парола се отваря личната база данни на съответния потребител.

Базата данни с потребителите (**users.db**) съдържа информация за наличните потребители в системата - потребителско име, парола и имейл.

Личната база данни на всеки потребител отговаря на потребителското име (потребител **ani123** → файл **ani123.db**) и съдържа информация за неговите пътувания - дестинация, период, оценка (от 1 до 5), коментар и множество от снимки (записани като имена на файлове).

Destination	Time period	Grade	Comment	Photos
Burgas, Bulgaria	15.07.2019 29.07.2019	5	A beautiful city on the Black Sea coast. I spent two unforgettable weeks there, meeting new people.	burgas.jpeg locumfest.png sunrise_on_the_coast.jpeg

Примерен запис от таблица ani123.db

Естествено, всеки иска да сподели информацията с приятелите си. Всеки потребител притежава и списък от приятели - други потребители на приложението. Той може да вижда дали те са посетили дадена дестинация и да вижда коментарите им за съответното пътуване.

Също така понякога потребителите на приложението ще желаят да проверят отзивите за дадена дестинация. Нека да могат да потърсят информация: да получат сведения кой потребител, посетил дестинацията, каква оценка е дал, както и средна оценка от всички.

Забележки:

- Нека дестинациите, за които има информация в приложението, бъдат изведени в списък. Ако някой потребител въведе нова дестинация, той да се актуализира.
- Нека има допълнителна валидация на информацията - да е невъзможно да се въведе невалиден период (освен валидни дати, крайната дата трябва да е след началната), да не може да се посочи невалиден файл като прикачена снимка (възможни разширения са jpeg и png, допустими са имена с малки и големи латински букви и символа _, без интервали) или невалидна оценка.
- Нека приложението има удобен за работа потребителски интерфейс.

Проект 8: Джурасик парк

Автор: Стела Маринова

Ванката беше изключително впечатлен от последната лекция в историческия музей и има нова хрумка - ще става палеонтолог! Не, още по-добре, ще си отвори ферма за динозаври!

Въпреки безумните убеждения на майка си, Ванката вече има план как ще изглежда новият домашен зоопарк за праисторически влечуги:

- динозаврите могат да са няколко вида:
    - месоядни
    - тревопасни
    - водни
    - летящи
- от книгите знае, че те не съжителстват заедно, затова трябва да са разделени
- необходимо е да има достатъчно храна за всички
- необходимо е да има достатъчно персонал в парка, за да се грижат за всички.

Вашата основна задача е да наподобите модела на успешен развъдник на динозаври.

Всеки динозавър се характеризира чрез

- име
- пол
- ера (триас, креда или юра)
- разред (тревопасен, месояден, летящ или воден)
- вид (плезиозавър, бронтозавър, тиранозавър и т.н.)
- храна (трева, месо или риба)

Те са разпределени в клетки, като заедно могат да съжителстват единствено животни от една ера и един разред (но не задължително от един вид). Съответно всяка клетка има подходящ климат - сухоземен, въздушен или воден. Има малки клетки (за едно животно), средни (за 3 животни) и големи (до 10 животни). Важна информация за клетката е:

- размер
- климат
- животни, които я обитават
- ера на животните вътре (ако има такива)

В нашия зоопарк отначало има произволно количество клетки и без никакви животни. Ново животно може да се приеме, ако има подходяща клетка за него с празно място в нея. Ако няма, то такава може да бъде построена, посочвайки съответния климат и размер.

Напишете система за управление на зоопарк, която може да:

- приема ново животно
- строи нова клетка
- премахва вече налично животно
- зарежда склада с храна

Забележки:

Може ли в началото клетките за динозаврите да са наистина случайно генерирани и като брой, и като вид?

Не искаме при затваряне на програмата информацията да се губи.

Летящите динозаври са в по-голямата си част хищни :)

Проект 9: Големи числа

(само за спец. Компютърни науки)

Автор: Траян Господинов

Всички сме чували за така наречените *големи числа* – това са числа, които теоретично не са ограничени отгоре. Има много начини за представянето на *големите числа*, като основния е символен низ и работата цифра по цифра (*в десетична бройна система*). Това представяне обаче е бавно и прахосващо памет, затова ще разгледаме представяне в *милиардична бройна система*. По този начин, като работим цифра по цифра, където цифрите са в *милиардична бройна система*, ще подобрим времето за изчисления деветократно.

Да се реализира:

Структура (клас) за работа с *големи числа*, представени в *милиардична бройна система*, която поддържа операциите: събиране, изваждане, умножение. Да бъдат спазени всички добри практики в ООП!

Упътване:

Цифра от *милиардичната бройна система* представлява променлива от тип, който поддържа числата, както и операциите с тях, в диапазона [0, 999999999].

Бонус:

Да се реализира бързо повдигане на степен на *големи числа* – от порядъка на O(n log n), където n е степента на която повдигаме. Резултата от повдигането на степен да бъде изведено по модул число от тип *unsigned long long*.

Проект 10: Библиотека

Да се напише програма, реализираща информационна система, която поддържа библиотека. Програмата съхранява и обработва данни за наличните в момента книги във файл.

Всяка книга се характеризира със следните данни:

- автор
- заглавие
- жанр
- кратко описание
- година на издаване
- ключови думи
- рейтинг
- уникален номер за библиотеката

Системата поддържа два вида потребители — администратори и клиенти на библиотеката. Всеки потребител се характеризира със следните данни:

- потребителско име
- парола
- ниво на достъп — указва дали потребителят е администратор или не.

След като приложението отвори даден файл, то трябва да може да извършва посочените по-долу операции, в допълнение на общите операции (`open, close, save, save as`, `help` и `exit`):

`login`	След въвеждането на командата потребителят последователно е питан за потребителско име и парола. Ако потребител с посочените данни съществува в програмата, се извежда съобщение “Welcome, <username>!”, където <username> съответства на потребителското име. В противен случай се извежда съобщение за грешно име или парола. При повторен опит за login, се изкарва съобщение “You are already logged in.”
`logout`	Потребителят напуска системата (програмата продължава да работи)
`books all`	извежда последователно за всяка книга следната информация:

	- заглавие, автор, жанр, персонален номер
`books info <isbn_value>`	извежда на екрана подробна информация за книга с персонален номер равен на <isbn_value> Пример: `books info 1124`

`books find <option> <option_string>`	`<option>` е едно от `title`, `author`, `tag`
	`<option_string>` е стойността на критерия за търсене, може да съдържа интервали
	Примери:
	`books find title Introduction to programming`
	`books find author Stephen King`
	`books find tag superhero`

`books sort <option> [asc | desc]`	`<option>` е едно от `title`, `author`, `year`, `rating`
	`asc` означава възходящо сортиране (по подразбиране), а `desc` означава низходящо сортиране
	Примери:
	`books sort title`
	`books sort year desc`

`users add <user> <password>`	Добавя нов потребител с потребителско име `<user>` и парола `<password>`. Потребителят и неговата парола се записват във файл.
`users remove`	Изтрива потребителя с потребителско име `<user>` от файла.

При първоначално стартиране на програмата няма налични данни за книги. Има регистриран по подразбиране само един потребител с администраторски акаунт със следните данни:

- потребителско име: `admin`
- парола: `i<3c++`

Програмата очаква да се въведе команда, като след въвеждането и се изпълнява според дефинираните правила. Това продължава до въвеждането на командата “`exit`”, която прекратява програмата.

Долната таблица описва за всяка от командите дали е достъпна само при коректно влязъл потребител и дали е ограничена само за потребителя `admin`.

команда	изисква ли потребител?	само за администратор?

open	не	не
close	не	не
save	не	не
saveas	не	не
help	не	не
login	не	не
logout	да	не
exit	не	не
books all	да	не
books find	да	не
books sort	да	не
books view	да	не
books add	да	да
books remove	да	да
users add	да	да
users remove	да	да

Бонус:

- при въвеждане на паролата на екрана да се изписва символа * вместо реалния символ
- при сортиране на книгите по зададен критерий, да се напише алгоритъм различен от пряка селекция и метода на мехурчето
- Търсене на книга по зададен критерий да игнорира регистъра на буквите (малки или големи)

Проект 11: Хотел

Да се напише програма, реализираща информационна система, обслужваща хотел. Програмата съхранява и обработва данните за стаите в хотела във файл.

След като приложението отвори даден файл, то трябва да може да извършва посочените по-долу операции, в допълнение на общите операции (`open, close, save, save as`, `help` и `exit`):

`checkin <room> <from> <to> <note>`	Регистриране в стая с номер `<room>` от дата `<from>` до дата `<to>` и се добавя бележка `<note>`. Пример: `checkin 229 2020-03-23 2020-03-31 The Simpsons`
`availability [<date>]`	Извежда списък на свободните стаи на дата `<date>`, ако не е зададена, се използва текущата дата.
`checkout <room>`	Освобождаване на заета стая с номер `<room>`.
`report <from> <to>`	Извежда справка за използването на стаи в периода от дата `<from>` до `<to>`. Извежда се списък, в който за всяка стая, използвана в дадения период, се извежда и броя на дните, в които е била използвана.
`find <beds> <from> <to>`	Намиране на подходяща свободна стая с поне `<beds>` на брой легла в периода от `<from>` до `<to>`. При наличие на повече свободни стаи се предпочитат такива с по-малко на брой легла.
`find! <beds> <from> <to>`	Да се реализира алгоритъм, който предлага спешно намиране на стая за важен гост в случай на липса на свободни стаи за даден период. Алгоритъмът да предлага разместване на настанените от най-много две стаи.
`unavailable <room> <from> <to> <note>`	Обявява стаята с номер `<room>` от дата `<from>` до дата `<to>` за временно недостъпна и се добавя бележка `<note>`. В стаята няма регистриран гост, но никой не може да бъде настанен в нея. Пример: `unavailable 200 2018-06-01 2019-03-01 Under construction`

Бонуси:

- гостите на хотела да могат да се записват по стая за различни дейности от даден списък при настаняване
    - да се извежда списък на програмата на една стая
    - да се извежда списък на всички записали се за дадена дейност

Проект 12: Склад

Да се напише програма, реализираща информационна система, обслужваща склад. Програмата съхранява и обработва данните за наличността в склада във файл.

За всеки продукт се съхранява следната информация:

- име (описание на продукта, символен низ с произволна дължина)
- срок на годност
- дата на постъпване в склада
- има не производител
- мерна единица (килограми, литри)
- налично количество
- местоположение (секция/рафт/номер)
    - номерирайте склада си както прецените, че ще ви е удобно, имайте предвид, че в началото той е празен и различно количество стока е нормално да заема различно по обем място
- коментар (свободен текст)

След като приложението отвори даден файл, то трябва да може да извършва посочените по-долу операции, в допълнение на общите операции (`open, close, save, save as`, `help` и `exit`):

`print`	Извежда информация за наличните продукти в склада
`add`	Добавя нов продукт в склада в диалогов режим, като въвежда всички данни за продукта и ги проверява за коректност. При добавянето на продукта се спазват следните правила: ● ако нов продукт е с различен срок на годност от вече съществуващ едноименен продукт, той да бъде поставен на различно място ● ако имате достатъчно място, еднакви продукти с един и същи срок на годност да бъдат поставени на едно и също място ● при извеждане на списъка с налични продукти да се изведе общото количество на едноименните продукти независимо от срока им на годност
`remove`	Изважда продукт от склада в диалогов режим: ● по дадено име и количество изважда съответните продукти от склада и извежда информацията за продукта и къде се е намирал ● при наличие на повече от една партида, първо намалява тази

	със най-скоро изтичащ срок на годност, тогава във информацията за извършеното действие се отбелязва количеството и мястото на всяка от партидите, които сме намалили ● в случай на опит за изваждане на повече от наличното да се дава информация на потребителя за наличността на продукта и срока на годност на партидите и възможност да реши дали все пак не иска да извади това което е останало
`log <from> <to>`	Извежда справка за всички промени в наличността в периода от дата `<from>` до дата `<to>`, включително зареждания и извеждания на стоки.
`clean`	Разчиства склада от всички стоки, на които е изтекъл или предстои скоро да изтече срока на годност, като извежда информация за разчистените стоки

Бонуси:

- по въведен продукт и неговата цена, за брой или съответно килограм, да се пресметнат загубите за даден от потребителя период (изхвърленото количество продукт със развален срок на годност)

Проект 13: СУСИ

Да се напише програма, реализираща информационна система за обслужване на студенти. Програмата съхранява и обработва необходимите данни във файл.

За всеки студент се съхранява следната информация:

- Име — символен низ с произволна дължина
- Факултетен номер
- Текущо записан курс, специалност, група
- статус (записан, прекъснал, завършил)
- Среден успех от следването до момента

Учебните дисциплини се характеризират с име и тип (задължителни / избираема).
За всяка специалност има списък от дисциплини и курс, в който могат да бъдат записвани, като за простота можете да приемете, че всяка дисциплина може да се записва само в рамките на точно един курс.

За всеки студент се пазят всички записани дисциплини и оценки по тези от тях, по които са положени изпити.

- добавяне на оценки/изпити (за даден предмет, оценка и студент)
- записване на задължителни/избираеми предмети
- отпечатване на протоколи
- академична справка за оценките на даден студент (списък с всички взети изпити ,съответните оценки и успех на студента). Да се включи също и списък с невзетите изпити (предмети, които са записани, но за тях няма оценки, те да се включат в пресмятането на средния успех на студента като двойки), ако има такива.

След като приложението отвори даден файл, то трябва да може да извършва посочените по-долу операции, в допълнение на общите операции (`open, close, save, save as`, `help` и `exit`):

`enroll <fn> <program> <group> <name>`	записване на студент с име `<name>` в 1 курс на специалност `<program>` в група `<group>` и с факултетен номер `<fn>`.
`advance <fn>`	записва студент в следващ курс
`change <fn> <option>`	`<option>` е едно от `program`, `group`, `year`. Прехвърля студента с факултетен номер `<fn>` в нова специалност

<value>	(program), група (group) или курс (year) зададени чрез <value>. Прехвърлянето в група е възможно винаги. Прехвърлянето в следващ курс е възможно само, ако студентът е положил успешно изпитите по всички задължителни предмети от минали курсове, с евентуално изключение на максимум два курса. Прехвърляне в курс, различен от следващия, не е възможно. Прехвърлянето в друга специалност е възможно само, ако студентът е положил успешно изпити по всички задължителни предмети от минали курсове на новата специалност.
graduate <fn>	Отбелязва студента като завършил, но само ако е положил успешно изпити по всички записани предмети.
interrupt <fn>	Маркира студента с факултетен номер <fn> като прекъснал. Прекъсналите студенти не могат да се явяват на изпити, да записват учебни дисциплини или да сменят специалност, група или курс.
resume <fn>	Възстановява студентските права на студента с факултетен номер <fn>.
print <fn>	Извежда справка за студента с факултетен номер <fn>.
printall <program> <year>	Извежда справка за всички студенти в дадена специалност и курс.
enrollin <fn> <course>	Записва студента с факултетен номер <fn> в дисциплината с име <course>. Записването е позволено само на дисциплини от съответния курс и специалност.
addgrade <fn> <course> <grade>	Добавя оценка <grade> по дисциплината <course> на студента с факултетен номер <fn>. Явяването на изпит е позволено само за дисциплини, които са записани.
protocol <course>	Извежда протоколи за всички студенти, записани в дадена дисциплина <course>. За всяка специалност и курс се извежда отделен протокол. Студентите в протокола са подредени по факултетен номер в нарастващ ред.
report <fn>	Извежда академична справка за оценките на даден студент (списък с всички взети изпити, съответните оценки и успех на студента). Да се включи също и списък с невзетите изпити (дисциплини, които са записани, но за тях няма оценки, те да се включат в пресмятането на средния успех на студента като двойки), ако има такива.

Бонуси:

- Да се поддържат дисциплини, които могат да бъдат записвани в няколко възможни курса (например 2, 3 и 4).
- Да се поддържат кредити за всяка избираема дисциплина и минимален брой за специалността и да се извежда справка колко още кредита от избираеми дисциплини трябва да вземе студента, за да може да се дипломира (кредитите от избираеми дисциплини, които студента не е взел, не се включват в текущите кредити).

Проект 14: Билети

Да се напише програма, реализираща информационна система, която обслужва билетна каса. Програмата съхранява и обработва необходимите данни във файл.

Представленията се играят в няколко зали, всяка от които има номер, брой редове и брой места на всеки ред. Залите са предварително зададени.

След като приложението отвори даден файл, то трябва да може да извършва посочените по-долу операции, в допълнение на общите операции (`open, close, save, save as`, `help` и `exit`):

`addevent <date> <hall> <name>`	Добавя ново представление на дата `<date>` с име `<name>` в зала `<hall>`. Ако в тази зала вече има друго представление на същата дата, командата да върне грешка.
`freeseats <date> <name>`	Извежда справка за свободните места за представление с име `<name>` на дата `<date>` (непродадени и незапазени билети).
`book <row> <seat> <date> <name> <note>`	Запазва билет за представление с име `<name>` на `<date>` на ред `<row>` и място `<seat>`, като добавя бележка `<note>`.
`unbook <row> <seat> <date> <name>`	Отменя резервация за представление с име `<name>` на `<date>` на ред `<row>` и място `<seat>`.
`buy <row> <seat> <date> <name>`	Закупува билет за представление с име `<name>` на `<date>` на ред `<row>` и място `<seat>`. За всеки билет се издава уникален сложен код, който съдържа информация за съответното място
`bookings [<date>] [<name>]`	Извежда справка за запазените, но неплатени (незакупени) билети за представление с име `<name>` на `<date>`. Ако `<name>` е пропуснато, извежда информация за всички представления на дадената дата. Ако `<date>` е пропуснато, извежда информация за всички дати.
`check <code>`	Прави проверка за валидност на билет, като по дадения код `<code>` се извлича номера на мястото или се връща грешка, ако кодът е невалиден).
`report <from>`	Извежда справка за закупени билети от дата `<from>` до дата `<to>` в

`<to> [<hall>]`	зала `<hall>`, като извежда всички изнесени представления в залата и за всяко отделно представление се извежда и количеството продадени билети. Ако `<hall>` е пропуснато, извежда информация за всички зали.

Бонуси:

- да се извежда статистика за най-гледаните представления
- да се извежда статистика за представления с под 10% посещаемост за даден период и да се дава възможност на потребителя да свали тези представления

Проект 15: Личен календар

Да се напише програма, реализираща информационна система, която поддържа личен календар, като го записва във файл.

След като приложението отвори даден файл, то трябва да може да извършва посочените по-долу операции, в допълнение на общите операции (`open, close, save, save as`, `help` и `exit`):

`book <date> <starttime> <endtime> <name> <note>`	Запазва час за среща с име `<name>` и коментар `<note>` на дата `<date>` с начален час `<starttime>` и краен час `<endtime>`.
`unbook <date> <starttime> <endtime>`	Отменя час за среща на дата `<date>` с начален час `<starttime>` и краен час `<endtime>`.
`agenda <date>`	Извежда хронологичен списък с всички ангажименти за деня `<date>`.
`change <date> <starttime> <option> <newvalue>`	`<option>` е едно от `date`, `starttime`, `enddate`, `name`, `note`. Задава нова стойност `<newvalue>` на събитието на дата `<date>` с начален час `<starttime>`, като при промяна на дата и час се прави проверка дали са коректни и свободни.
`find <string>`	Търсене на среща: извеждат се данните за всички срещи, в чието име или бележка се съдържа низът `<string>`.
`holiday <date>`	Датата `<date>` се отбелязва като неработна.
`busydays <from> <to>`	Извеждане на статистика за натовареност: по дадени начална дата `<from>` и крайна дата `<to>` се извежда списък с дните от седмицата, подредени по критерия “брой заети часове”.
`findslot <fromdate> <hours>`	Намиране на свободно място за среща: по дадена дата `<fromdate>` и желана продължителност на срещата `<hours>` търси дата, на която е възможно да се запази такава среща, но само в работни дни и не преди 8 часа или след 17 часа.
`findslotwith <fromdate>`	Намиране на свободно място за среща, синхронизирана с даден календар: по дадена дата `<fromdate>` и желана продължителност на

`<hours> <calendar>`	срещата `<hours>` търси дата, на която е възможно да се запази такава среща в текущия календар и в календара, записан във файл `<calendar>`, но само в работни дни и не преди 8 часа или след 17 часа.
`merge <calendar>`	Прехвърля всички събития от календара, записан във файл <calendar>, в текущия календар. Прехвърлянето да става в диалогов режим така, че ако има конфликт на събития потребителят да има възможност да избере кое събитие да остане и кое да се премести в друг ден и час. Пример: потребителят се е записал на спорт и е получил файл, който съдържа календар с всички тренировки и спортни събития. Той иска да прехвърли всички спортни събития в календара си.

Бонуси:

- командите `findslotwith` и `merge` да поддържат повече от един календар.

Проект 16: JSON парсер

JSON е популярен текстов формат за описване на данни. Да се напише програма, която работи с файлове в такъв формат.

След като приложението отвори даден файл, то трябва да може да извършва посочените по-долу операции, в допълнение на общите операции (`open, close, save, save as, help` и `exit`):

`validate`	Да се направи проверка дали отворения файл е валиден спрямо синтаксиса на JSON. Ако има някакъв проблем, програмата трябва да съобщи максимално ясно какво и къде не е наред, така че потребителят да може да го поправи.
`print`	Да се изведе съдържанието на обекта в максимално четим вид.
`search <key>`	Да се провери дали в обекта се съдържат данни, записани под този ключ и ако да – да изведе списък от всички такива данни.
`set <path> <string>`	При подаване на пълен път `<path>` към даден елемент, да се замени стойността на посочения елемент с нов обект, получен от низа `<string>` според синтаксиса на JSON стойност, ако такъв елемент съществува и е единствен и ако символният низ е коректен. В противен случай трябва да обяви каква е грешката.
`create <path> <string>`	При подаване на пълен път `<path>` към даден елемент да се добави такъв елемент, получен от низа `<string>` според синтаксиса на JSON стойност. Ако такъв елемент съществува, това трябва да се обяви като грешка. В случай, че последните елементи от пътя не съществуват, да се създадат. Ако символният низ или пътят не са коректни, да се съобщи с подходящо описание на грешката.
`delete <path>`	При подаване на пълен път <path> до елемент, да се изтрие, ако такъв съществува или да съобщи на потребителя при некоректен път.
`move <from> <to>`	Всички елементи, намиращи се на път `<from>` да бъдат преместени на пътя `<to>`.
`save [<path>]` `saveas <file> [<path>]`	Командите save и saves да работят с произволен път, като записват в текущия или в нов файл обекта на дадения път, ако съществува. Ако

	<path> не е подаден, да се записва целият обект, който в момента е зареден в паметта.

Проект 17: Растерна графика

В рамките на този проект трябва да се разработи приложение, което представлява конзолен редактор на растерни изображения. Вашия редактор трябва да може да поддържа работа с различни файлове, стартиране на сесия/и, прилагане на различни трансформации върху изображенията и разбира се записване на резултата.

Като минимум приложението ви трябва да може да работи с PPM, PGM и PBM файлове (за повече информация вижте http://en.wikipedia.org/wiki/Netpbm_format).

Зареждането на файл с командата `load` създава "потребителска сесия", която има уникален номер. В рамките на една сесия могат да са заредени повече от едно изображение. Изпълнението на командата `load` създава нова потребителска сесия. След командата load се очаква поне едно име на файл, което да се зареди със създаването на сесията. Все пак сесия без снимки в нея е безсмислено.

Пример:
```
> load lolcat.ppm
Session with ID: 1 started
```

Програмата трябва да дава възможност за изпълняване на различни трансформации върху файловете в дадена сесия. Когато се прилагат трансформации в сесия, то те важат за всички заредени изображения за **текущата** сесия.

Трансформациите се прилагат над изображенията едва след като се изпълни команда `save` или `saveas`. Командата `save` записва всички заредени в текущата потребителска сесия изображения след като прилага всички трансформации, а командата `saveas` записва под ново име само изображението, което е заредено първо.

След като приложението отвори даден файл, то трябва да може да извършва посочените по-долу операции, в допълнение на общите операции (`load, close, save, save as, help` и `exit`):

`grayscale`	При извикване на тази команда, тя трябва да се приложи върху всички цветни изображения в текущата сесия. Ако в текущата сесия са включени черно-бели изображения, то те не трябва да бъдат модифицирани. Обърнете внимание, че един файл може да бъде във формат, който поддържа цветни изображения (например PPM), но да бъде чернобял (т.е. всички пиксели в него са нюанси на сивото).
`monochrome`	Поведението на тази операция е като това на предишната, но тук изображението се преобразува до монохромно (такова, в което има само черни и бели пиксели, без никакви нюанси на сивото). Отново,

	ако изображението вече е монохромно, тази трансформация не го променя.
`negative`	Тази команда прави негатив (цветово обръщане) на изображенията в текущата сесия.
`rotate <direction>`	`<direction>` е едно от `left` и `right` Реализира се завъртане на 90 градуса в съответната посока.
`undo`	Като един истински редактор за изображения, приложението трябва да поддържа и команда за отмяна на действие**,** която премахва последно направената трансформация в текущата сесия. Ако е стартирана нова сесия и след това веднага бъде въведена команда `undo`, то тя не трябва да има никакъв ефект.
`add <image>`	Добавя изображението `<image>` към текущата сесия. Всички приложени до момента трансформации не се прилагат върху него.
`session info`	Дава възможност на потребителя да получава подробна информация за текущата потребителска сесия нейният идентификационен номер, участващите изображения, както и набора от трансформации, които до този момент предстоят да бъдат приложени върху съответните изображения, участващи в сесията.
`switch <session>`	Превключва към сесия с идентификационен номер `<session>`. Ако сесия с такъв номер не съществува, да се изведе подходящо съобщение за грешка.
`collage <direction> <image1> <image2> <outimage>`	<direction> е едно от `horizontal` и `vertical` Създава колаж от две изображения `<image1>` и `<image2>` (в един и същ формат и една и съща размерност), налични в текущата сесия. Резултатът се записва в ново изображение `<outimage>`, което се добавя в текущата сесия.

Пример 1:

```
> load image1.ppm
Session with ID: 1 started
Image "image1.ppm" added

> add image2.pgm
Image "image2.pgm" added

> add image3.pbm
Image "image2.pgm" added
```

```
> rotate left

> grayscale

> session info
Name of images in the session: img1.ppm img2.pgm
Pending transformations: rotate left, grayscale
```

Пример 2:

```
> load img1.ppm img2.pbm
Session with ID: 1 started
Image "img1.ppm" added
Image "img2.pbm" added

> add img3.pgm
Image "img3.ppm" added

> grayscale

> rotate left

> load img4.pbm
Session with ID: 2 started
Image "img4.pbm" added

> add img5.ppm
Image "img5.ppm" added

> rotate right

> switch 1
You switched to session with ID: 1!
Name of images in the session: img1.ppm img2.pbm img3.pgm
Pending transformations: grayscale, rotate left

> switch 2
You switched to session with ID: 2!
Name of images in the session: img4.pbm img5.ppm
Pending transformations: rotate right
```

Пример 3:

```
> load image1.ppm
Session with ID: 1 started
Image "image1.ppm" added

> add image2.ppm
Image "image2.ppm" added

> add image3.pgm
Image "image3.pgm" added

> collage horizontal image1.ppm image2.ppm collage.ppm
New collage "collage.ppm" created

> collage vertical image1.ppm image3.pgm
Cannot make a collage from different types! (.ppm and .pgm)

> save
```

Проект 18: Dungeons & Dragons

Автор: Стела Маринова

Проектирайте игра, подобна на известната “Dungeons and Dragons” (D&D). Основната цел на играта е вашият герой да се придвижва по игрално поле - карта, събирайки съкровища и побеждавайки чудовища.

Игрално поле

Игралното поле представлява карта на лабиринт, като героят се позиционира в началото в горния ляв ъгъл и трябва да достигне долния десен ъгъл, за да завърши успешно, движейки се нагоре, надолу, наляво и надясно. Стените са отбелязани със знака **#**, а свободните полета - с **.**. По пътя са разположени съкровища (**T**) и чудовища (**M**), разположени на случаен принцип.

Герой

Героят на играча има раса, към която принадлежи, и която определя стартовото отношение на трите основни показателя:

- **сила** - това е грубата сила на героя в битка
- **мана** - силата на героя при извършване на заклинание
- **здраве** - физическото здраве на героя. Достигне ли 0, героят умира

Раса	Сила	Мана	Здраве
човек	30	20	50
маг	10	40	50
воин	40	10	50

С повишаване на нивото, всеки герой получава още 30 точки, които може да разпредели между показателите си.

Също така всеки герой притежава личен инвентар:

- **броня** - предпазва от атака, намалявайки я с определен процент
- **оръжие** - добавя процент към силовата атака на героя
- **заклинание** - добавя процент към атаката със заклинание

Всеки герой може да притежава по един предмет от всеки тип, като при откриване на нов предмет, решава дали да го екипира и да замени стария, или да го изхвърли.

В началото при регистрация в играта, героят е на ниво 1. Инвентарът му се състои от обикновен меч (20%) и огнена топка (20%), без броня.

Съкровища

Съкровищата в играта са предмети - броня, оръжие или заклинание. В началото на играта, те се позиционират на игралното поле, избирайки се на случаен принцип от списък, подходящ за нивото. Ако героят попадне на поле със съкровище, той се изправя пред избора да го задържи или изхвърли.

Чудовища

В играта има също и чудовища - дракони, притежаващи показатели в съответствие с нивото. В началото на играта (на първо ниво) те са:

Сила	Мана	Здраве
25	25	50

Също така люспите им служат за броня и намалят атаката с 15%.

Те стоят неподвижно там, където са позиционирани в началото на играта. При среща на героя с тях (попадане на поле, на което има дракон), настъпва битка.

Битка

Ако настъпи битка, героят и чудовището се изправят един срещу друг. На случаен принцип се определя дали героят или чудовището започват битката. На всеки ход героят решава дали да използва силова атака или да възпроизведе заклинание. Чудовището избира между двете опции на случаен принцип. Съответно здравето на героя/чудовището се намалява с толкова точки, колкото е силата на атаката (след прилагането на бонуса от оръжие/заклинание и неутрализацията на бронята). Ако здравето на чудовището достигне 0, то героят е победител, неговото здраве се възстановява на 50% от първоначалното и той продължава играта. Ако здравето на героя достигне 0, то той умира и играта приключва.

Нива

С напредване на играта, т.е. при успешно преминаване на лабиринт, героят вдига нивото си, а следващата генерирана карта, заедно с чудовищата и съкровищата, прилежащи към нея, отговарят на това ново ниво.

При преминаване на следващо ниво, героят получава 30 точки, които има право да разпредели между трите си показателя.

С покачване на нивото, чудовищата получават по 10 точки към всеки един свой показател и 5% към бронята си.

Вашата задача

Създайте прототип на D&D. Размерите на картата, броя на чудовища и съкровища се задават по следния начин:

ниво 1: карта 10x10, 2 чудовища, 2 съкровища.
ниво 2: карта 15x10, 3 чудовища, 2 съкровища.
всяко следващо ниво се определя като сума на показателите от предишните две:
ниво 3: карта 25x20, 5 чудовища, 4 съкровища.
ниво 4: карта 40x30, 8 чудовища, 6 съкровища
и т.н.

Нека информацията за картата на съответното ниво и списъка с прилежащи съкровища се чете от файл.

Интересни моменти:

- Какво става, ако здравето на героя след битка е повече от 50%? Трябва ли да намалява в такъв случай?
- Предметите трябва да отговарят на нивото. Какви са подходящите граници за всяко ниво? Оптимално ли е още на първо ниво да са достъпни много силни оръжия?
- Може ли картата и списъкът със съкровища да се генерират на случаен принцип чрез изпълнение на команда **generate_level**? Може ли да идва администраторски режим, позволяващ това? Как става достъпването му?
- Играта трябва да е интересна и лесна за потребителите. Какъв е подходящият интерфейс за това?

Проект 19: Star Wars Universe 0.1

(само за спец. Компютърни науки)

Автор: Георги Шавов

Ванката е голям фен на Star Wars, затова решил да си направи малко проектче по ООП на тази тематика. За целта той иска да пресъздаде вселената от поредицата.

Първо той започнал с джедаите, като решил че всеки такъв трябва да притежава:

- джедайско име
- ранг, като следните са подредени в нарастващ ред – YOUNGLING, INITIATE, PADAWAN, KNIGHT-ASPIRANT, KNIGHT, MASTER, BATTLE_MASTER и GRAND_MASTER
- възраст
- цвят на светлинния меч (символен низ, който се въвежда от клавиатурата)
- сила (зададена с някакво число от тип double)

След това решил да създаде планетите и луните, като всяка такава има име и джедаи, които я населяват.

Да се изготви приложение, което поддържа следните команди:

`add_planet <planet_name>`	добавя нова планета
`create_jedi <planet_name> <jedi_name> <jedi_rank> <jedi_age> <saber_color> <jedi_strength>`	функцията да извежда съобщение дали добавянето е било успешно или не (съществува джедай с такова име на тази или друга планета, или не съществува планета с такова име).
`removeJedi <jedi_name> <planet_name>`	функцията да извежда съобщение дали премахването е било успешно или не (джедаят не населява тази планета).

`promote_jedi <jedi_name> <multiplier>`	повишава дадения джедай с един ранг нагоре в стълбицата и увеличава силата му по формулата `jedi_strength += (multiplier * jedi_strength)` (не може да се повишава повече от ранг `GRAND_MASTER` и `multiplier` трябва да е положително число от тип `double`)
`demote_jedi <jedi_name> <multiplier>`	намаля ранга на подаденият джедай с един ранг надолу в стълбицата и понижава силата му по формулата `jedi_strength -= (multiplier * jedi_strength)` (не може да се понижава повече от ранг `YOUNGLING` и `multiplier` трябва да е положително число от тип `double`)
`get_strongest_jedi <planet_name>`	извежда информацията за най-силния джедай на подадената планета (с най-голяма сила).
`get_youngest_jedi <planet_name> <jedi_rank>`	извежда най-младия джедай, населяващ подадената планета и е със съответен ранг (**ако са повече от един, да се изведе първият по азбучен ред, ако няма нито един да се изведе подходящо съобщение**)
`get_most_used_saber_color <planet_name> <jedi_rank>`	връща най-разпространения цвят на светлинния меч в подадения ранг на съответната планета
`get_most_used_saber_color <planet_name>`	връща най-разпространения цвят на светлинния меч планета, който се ползва от поне един **GRAND_MASTER**
`print <planet_name>`	извежда по подходящ начин името на планетата и населяващите я джедаи, **сортирани първо в нарастващ ред по ранг, после по втори ключ - лексикографски по името**

`print <jedi_name>`	извежда по подходящ начин информацията за джедаят, както и коя планета населява в момента
`<planet_name> + <planet_name>`	извежда на екрана в сортиран вид (**лексикографски**) информацията за населяващите двете планети джедаи.

Ванката искал освен това да може да запазва информацията, която е вкарал в програмата си за после и да може да я зарежда наново. Тоест програмата трябва да съхранява информацията планетите и населяващите ги джедаи **във файл** и да се поддържат командите за работа с файлове, описани в секцията **Работа с командния ред**. **Всички команди са с малки латински букви, а аргументите са разделени с един интервал.**

Проект 20: Шах

Автор: Иван Арабаджийски

Задачата ви е да напишете програма, която позволява на двама играчи да провеждат стандартна игра на шах.

Играта ще се провежда чрез команди на конзолата.

Функционалности:

Клас 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 поле по диагонал от нея

Бонус: да се поддържа правилото en passant

- **Топ** (Rook) : може да се придвижва хоризонтално или вертикално до достъпните квадратчета и да атакува фигури, намиращи се по тях.

Бонус: да се поддържа възможност за рокада

- **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; може да се реализира чрез наследяване на тези 2 класа;

- **Цар** (King) : може да се придвижва и атакува само фигури само в непосредствено съседство;

Бонус: да се поддържа възможност за рокада

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

● **canMove()** - 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.

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

Пример: Всяко квадратче от дъската може да бъде клас, съдържащ координатите си и фигурата, която стои върху него. В такъв случай дъската би била матрица от полета.

Клас Игра (Game) : 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);

Уточнение: Класът Game трябва да има метод move, но аргументите, които този метод приема не е задължително да са в показаният горе формат, стига функционалността да е спазена. Той е само пример за ваше улеснение.

● **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.

Да се изготви приложение, което поддържа следните команди:

● **move xx yy** - премества фигура от поле xx на поле yy, където xx и yy са полета от дъската

Пример: move d2 d4

Бонус: да се поддържа възможност за рокада

● **help** - дава информация за командите на играчите

● **undo** - връща играта в състоянието преди последния ход.

● **exit** - прекратява играта преждевременно

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.

Забележка: Стига изисканите функционалности да са спазени, реализациите могат да варират, дадените в условието са само примерни за ваше улеснение. Единствено стриктно трябва да бъдат спазени синтактично командите в секция “Да се изготви приложение, което поддържа следните команди”.

Бонус проекти: игри

Този раздел съдържа проекти, които реализират някои популярни компютърни игри от миналия век. Не е нужно игрите да бъдат реализирани в графична среда, нито да използват други входни устройства (мишка, gamepad или други). Приемат се решения реализирани дори само с cin и cout, но използването на допълнителни библиотеки за постигане на по-добри ефекти е позволено. Вместо подробно описание на игрите, към повечето от тях е добавена връзка, на която можете сами да поиграете и да получите представа за логиката на играта.

Общи бонуси за всички игри:

- графика
- звук
- запазване на най-високите резултати (high score) при спиране на играта
- запазване на състоянието на играта (save) с възможност за продължаване при рестартиране
- всякакви ваши оригинални идеи :)

Snake

Примерна игра: http://goldfirestudios.com/proj/snake/

Бонуси:

- възможност за двама играчи
- възможност компютърът да управлява автоматично една от змиите
- различни нива с препятствия
- “развалени” хапчета, които скъсяват змията

Alien Attack

Примерна игра (изисква Flash): http://www.mysteinbach.ca/game-zone/85/space-invaders/

Бонуси:

- възможност за двама играчи
- възможност компютърът да управлява автоматично единия кораб
- бонуси, които играчът може да събира (живот, бомби, щитове, реконструиране на крепости)

Sokoban

Примерна игра: http://www.game-sokoban.com/

Бонуси:

- автоматично генериране на нива
- подсказване от компютъра
- бонуси, които играчът може да събира (кука за дърпане, вместо само за бутане, бутане на две квадратчето едновременно, бомба за стени)

Tetris

Примерна игра: http://www.htmltetris.com/

Бонуси:

- подсказване от компютъра (къде е най-добре да се сложи фигурата)
- режим “късметлия” — да се пускат най-удобните в дадена ситуация фигури
- режим “карък” — да се пускат най-неудобните в дадена ситуация фигури

Breakout

Примерна игра: https://www.gamescaptain.com/play_online_game/breakout

Бонуси:

- специални тухли (чупещи се по-трудно, взривяващи се, нечупливи, телепортиращи, изчезващи)
- бонуси за играча (животи, стрелба, повече топчета, топчета с по-голяма сила, удължаване/скъсяване на плочката, лепкава плочка)
- автоматична игра на компютъра

Xonix

Примерна игра: http://www.youtube.com/watch?v=POhMfAFZ_6c

Бонуси:

- “умни” врагове, които преследват топчето
- бонуси за играча (временно безсмъртие, стрелба, животи)

Pacman

Примерна игра: http://pacman.platzh1rsch.ch/

Бонуси:

- “умни” духчета, които преследват играча
- генериране на произволни лабиринти

Mine Sweeper

Примерна игра: http://minesweeperonline.com/

Бонуси:

- “жокери” (отваряне на произволно квадратче без мина)
- избор на игра с произволни параметри (брой мини, размер)

Pong

Примерна игра: https://www.retrogames.cz/play_530-DOS.php

Бонуси:

- компютърът като автоматичен играч
- забързване на топчето при удар “в движение”

Lander

Примерна игра: http://moonlander.seb.ly/

Бонуси:

- генериране на случайни терени

- симулиране на различна гравитация за различни планети
- двама играчи едновременно (кой по-бързо ще приземи)
- компютърът като автоматичен играч

Arcade Volleyball

Примерни игри:
http://blobby.sourceforge.net/data/bv2browser/

Бонуси:
- компютърът като автоматичен играч
- забавяне на топката при докосване на мрежата
- подскачане на различна височина (в зависимост от продължителността на натискане на клавиша за скок)

Frogger

Примерна игра: https://www.froggergames.net/games/html5frogger.html

Бонуси:
- възможност за двама играчи
- компютърът като автоматичен играч
- режим “късметлия” — минаващите коли се опитват да не блъснат животното
- режим “карък” — минаващите коли се опитват да блъснат животното

The Game of Life

(вижте повече в Wikipedia на английски или български)

Да се напише програма, която реализира “Игра на Живот”. В Играта на Живот има "дъска" състояща се от M x N квадратни клетки (10 ≤ M, N ≤ 60), дори да е възможна с по-малки стойности от дадените играта е безинтересна). Във всяка клетка или има живо същество или не. Две клетки наричаме съседни, ако имат обща стена или ъгъл. Живите същества могат да бъдат от различни племена (т.е. условно можем да ги наречем Варвари, Рицари, Граждани и т.н). Всяко племе си има собствен “знак” (т.е ‘%’ за Варварин, “+” за Рицар, “&” за Гражданин и т.н). Дъската има начално състояние. То определя се в кои клетки има живот, в кои не и кое същество от кое племе е. Играта се състои от безброй много ходове,

като на всеки ход дъската се променя по следните правила:

- Всяка жива клетка с по-малко от две живи (независимо от племето) съседни клетки умира (от самота).
- Всяка жива клетка с повече от три живи (независимо от племето) съседни клетки умира (от пренаселеност).
- Всяка жива клетка с две или три живи (от същото племе) съседни клетки остава жива и на следващата итерация.
- Съдбата на всяка жива клетка, която не удовлетворява никое горните условия, се решава от случайността (шанс 50% за живот и 50% за смърт).
- Всяка мъртва клетка с точно три живи (от същото племе) съседни клетки се превръща в жива клетка (от съответното племе).

След края на всеки ход дъската трябва да се визуализира на екрана. Между два последователни хода трябва да се изчака известно време за да може човек да наблюдава развитието на играта.

Бонуси:

- Всеки път дъската се инициализира с произволен размер, брой племена и състояние на клетките.
- Опция за игра до “победа”. Като (някои) от избираемите условия за победа могат да са:
  - Да остане само едно “живо” племе
  - Да умрат всички племена
  - Дъската да не се е променила в рамките на два последователни хода
- Да се засичат ситуации, в които никой не печели, понеже дадено състояние на дъската започва да се повтаря винаги, независимо от случайността.

2048

Оригиналната игра: http://gabrielecirulli.github.io/2048/

Бонуси:

- да се реализира възможност за undo/redo (без да се променя позицията и стойността на случайно появяващите се клетки)
- режим “късметлия” — появяващите се клетки да са максимално изгодни за играча
- режим “карък” — появяващите се клетки да са максимално неудобни за играча

Занимателна математика

Да се напише програма която реализира играта “Занимателна математика”, чиито

правила са описани тук.

Играта да позволява възможност двама редуващи се играчи, както и на играч срещу компютъра. Плочките за игра могат да са случайно генерирани числа в интервала [-10; 10]. Плочките в оригиналната игра са както следва: 1 x -10, 1 x -7, 2 x -6, 2 x -5, 3 x -4, 3 x -3, 4 x -2, 5 x -1, 4 x 0, 5 x 1, 8 x 2, 5 x 3, 4 x 4, 4 x 5, 5 x 6, 3 x 7, 2 x 8, 1 x 10, 1 x 15 (общо 63 на брой, без да се брои плочката ☺)

Проект 18: Dungeons & Dragons

Вижте описанието по-горе

Проект 20: Шах

Вижте описанието по-горе