Работа с командния ред

Следните условия за работа с команден ред **важат за всички теми за проекти**. Моля, запознайте се внимателно с тях.

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 (`saveas`). Трябва да има и опция за затваряне на файла без записване на промените (`close`).

За да предостави на потребителя възможност да укаже каква операция да се изпълни, програмата трябва да работи в **интерактивен команден режим**: при стартиране, тя подканва потребителя да въведе едноредови команди и след това в зависимост от въведените команди да ги изпълнява. Командите могат да имат нула, един или повече параметри, които се изреждат един след друг, разделени с един или няколко интервала. Някои от командите може да изискват допълнително въвеждане на информация при изпълнението си.

При отваряне на 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. При изход, програмата трябва да подсеща потребителя дали желае да запише последните промени във файла. Подсещането да се извежда само ако след последното записване е изпълнена операция, която променя данните в паметта.

Бонус: Ако желаете, можете да реализирате възможност, при която текущото състояние на паметта се записва във временен файл за възстановяване (recovery file) като защита при евентуално прекратяване на програмата при грешка. При отваряне на файл, програмата да проверява за наличие на файл за възстановяване и да предлага на потребителя да избере дали данните от него да бъдат възстановени или пренебрегнати.

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

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

Освен ако не е указано друго, всяка от командите следва да 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.

След като файлът бъде отворен и се прочете, той се затваря и програмата вече не трябва да работи с него, освен ако потребителят не поиска да запише обратно направените промени (както е указано в командите **Save** и **Save As** по-долу), в който случай файлът трябва да се отвори отново за запис.

Ако при зареждането на данните, програмата открие грешка, тя трябва да изведе подходящо съобщение за грешка и да прекрати своето изпълнение.

Пример:

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

Successfully opened file.xml
```

Close

Затваря текущо отворения файл. Командата се изпълнява успешно само ако има текущ отворен файл. Затварянето изчиства текущо заредената информация и след това програмата не може да изпълнява други команди, освен отваряне на файл (Open).

Пример:

```
> close

Successfully closed file.xml
```

Save

Записва направените промени обратно в същия файл, от който са били прочетени данните. Командата се изпълнява успешно само ако има текущ отворен файл.

Пример:

```
> save

Successfully saved file.xml
```

Save As

Записва направените промени във файл, като позволява на потребителя да укаже неговия път. Командата се изпълнява успешно само ако има текущ отворен файл.

Пример:

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

Successfully saved another file.xml
```

Help

Извежда кратка информация за поддържаните от програмата команди.

Пример:

```
> 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. Ако има незаписани промени, предлага на потребителя да избере затваряне с пренебрегване на промените (Close) или записване (Save или Save As).

```
> exit

You have an open file with unsaved changes, please select close or save
first.

> close

Successfully closed file.xml

> exit

Exiting program...
```

Тема 7: 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. Програмата трябва 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 – поредица от произволни символи оградени в кавички (quotation marks). Подобно на низовете в C++, включването на символа за кавичка в даден низ, трябва да се представи като `\"`, а включването на обратна наклонена черта се представя 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`, `saveas`, `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.

Ако при зареждането на данните се открие грешка (напр. некоректен формат), трябва да се изведе подходящо съобщение за грешка и изпълнението на програмата да се прекрати. 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 има подходящи параметри и да позволява 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 данни от произволен тип от гореописаните.

Ако потребителят въведе неправилни данни, прогрмата не трябва да променя нищо в таблицата, а само да изведе на екрана съобщение, че са въведени неправилни данни. В този случай програмата 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 други клетки от произволни типове.

При изчислението на стойността на дадена клетка важат следните правила:

1. Ако в дадена формула участват само числа, то пресмятането се извършва по традиционните правила на аритметиката като се спазва приоритетът на операциите. Като специален случай можем да отделим делението на две цели числа: резултатът може да е дробно число (например 1 делено на 2 дава резултат 0.5).
2. Ако в дадена формула участва низ, който коректно представя цяло или дробно число съгласно горното описание, той трябва да се преобразува до стойността на числото Всички други низове, участващи във формула, се конвертират до нула. Например:

Низ	Стойност във формула
"123"	123
"123.456.789"	0
"123.456"	123.456
"Hello world"	0
"123abc"	0

3. Ако в дадена формула участва празна клетка, нейната стойност се счита за нула. Това важи и за клетки, чиито координати надхвърлят размерите на таблицата.
4. Ако в дадена формула има грешка (например деление на нула), програмата 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, а 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 примерни формули, които се оценяват в примерната таблица по-горе. За всяка формула е дадена и нейната оценка:

Формула	Пресмятане	Стойност	Коментар
= 10 + 10	10 + 10	20	
= R1C1 + R1C3	10 + 123.56	133.56	
= R1C1 * R1C2	10 * 0	0	Стойността на низа "Hello world!" се счита за 0.
= R1C1 * R2C1	10 * 123	1230	Стойността на низа "123" е 123.
= R1C1 * R2C2	10 * 0	0	Клетката на ред 2, колона 2 е празна.
= R1C1 * R200C1	10 * 0	0	В таблицата няма ред 200, считаме, че тя е празна.
= 10 / 0	10 / 0	ERROR	
= 10 / R2C2	10 / 0	ERROR	Клетката на ред 2, колона 2 е празна.
= R1C1 / R1C2	10 / 0	ERROR	В таблицата няма ред 200 и колона 200. Считаме, че тя е празна.

Тема 8: SVG файлове

В рамките на този проект трябва да се разработи програма, която работи със файлове, представени чрез подмножество на формата Scalable Vector Graphics (SVG). Програмата 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 програмата трябва да е такъв, че да позволява при нужда лесно да може да се добави поддръжка на нови фигури.

Когато съдържанието на един SVG файл се зареди, трябва да се прочетат само фигурите, които програмата поддържа, всички останали SVG елементи могат да се игнорират.

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

Записаният от програмата файл трябва да е валиден файл във формата SVG, който да може коректно да се отваря от други програми (напр. Браузър).

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`, `saveas`, `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`)

Пример:

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
```

Тема 9: XML парсер

Да се реализира 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, за идентификатор да се ползва тази стойност, към която е слепен подходящ низ, който да допълни идентификатора до уникален такъв (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, saveas, 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, вложените 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"` връща списък с всички адреси във файла)
- ∉ да поддържат оператора `[]` (например `"person/address[0]"` връща адреса на първия елемент във файла)
- ∉ да поддържат оператора `@` (например `"person(@id)"` дава списък с атрибутите `id` на всички елементи във файла)
- ∉ Оператори за сравнение `=` (например `"person(address='USA')/name"` дава списък с имената на всички елементи, чиито адреси са точно равни на `"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 имат основните характеристики на XML (както файла в горния пример, който всъщност не е валиден XML), а завките да имат основните характеристики на XPath.

Бонуси:

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

Тема 10: Бази от данни

Да се реализира програма, поддържаща операции с прости бази от данни. Базите данни се състоят от множество 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 – поредица от произволни символи оградени в кавички (quotation marks). Подобно на низовете в C++, включването на символа за кавичка в даден низ, трябва да се представи като `\"`, а включването на обратна наклонена черта се представя 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`, `saveas`, `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, програмата да

	извжеда 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 с пореден номер `<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 n1> <table 2> <column n2>`	Извършва операцията Inner Join над две таблици спрямо колоните с поредни номера `<column n1>` в първата таблица и `<column n2>` във втората. Създава нова таблица с автоматично генериран идентификатор и го извежда
`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 с пореден номер `<target column n>` на всички редове, чиито колони с номер `<search column n>` съдържат стойността `<search value>`. Възможните стойности на `<operation>` са `sum`, `product`, `maximum`, `minimum`. Системата да дава грешка, ако колоната `<target column n>` не е от числов тип.

Тема 11: Растерна графика

В рамките на този проект трябва да се разработи програма, която представлява конзолен редактор на растерни изображения. Редакторът трябва да поддържа работа с различни файлови формати, работа с множество потребителски сесии и прилагане на различни трансформации върху изображенията.

Като минимум, програмата 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 `open` създава "потребителска сесия", която има уникален номер. **В рамките на една сесия могат да са заредени повече от едно изображение.** Изпълнението на командата `open` създава нова потребителска сесия. След командата `open` се очаква поне едно име на файл, което да се зареди със създаването на сесията.

Пример:

```
> open 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`, `saveas`, `help` и `exit`):

`grayscale`	Тази команда прилага преобразуване всички цветни изображения в текущата сесия до такива, използващи нюанси на сивото. 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 прилага трансформацията негатив (цветово обръщане) на изображенията в текущата сесия.
`rotate <direction>`	`<direction>` е едно от `left` и `right` Тази команда прилага 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

Тема 12: Star Wars Universe 0.1

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 реализира програма, която поддържа следните команди:

`add_planet <planet_name>`	Добавя нова планета
`create_jedi <planet_name> <jedi_name> <jedi_rank> <jedi_age> <saber_color> <jedi_strength>`	Създава нов джедай с подадените параметри. Извежда съобщение дали добавянето е било успешно или не (напр. съществува джедай с такова име на тази или друга планета или не съществува планета с такова име)
`removeJedi <jedi_name> <planet_name>`	Премахва съществуващ джедай. Извежда съобщение дали премахването е било успешно или не (напр. 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 *= (1 + multiplier)` (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>`	Намалява ранга на подаденият джедай с един ранг надолу в стълбицата и понижава силата му по формулата `jedi_strength *= (1 - multiplier)` (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, а равните по ранг – по азбученред на имената**
`print <jedi_name>`	Извежда по подходящ начин информацията за джедая, както и коя планета населява в момента
`<planet_name> + <planet_name>`	Извежда на екрана информацията за населяващите двете планети джедаи, сортирани по азбучен ред на имената.

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.**

Тема 13: Dungeons & Dragons

Да се проектира, подобна на известната “Dungeons and Dragons” (D&D). Основната цел на играта е героят 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 – със знака точка (**.**). 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

Стартовите показатели се определят съгласно следната таблица:

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 по свое усмотрение.

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 на атакуващия и последваща неутрализацията, причинена от бронята на защитаващия се). 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 играта D&D. Размерите на картата, броя на чудовища и съкровища се задават по следния начин:

ниво 1: карта 10 × 10, 2 чудовища, 2 съкровища.

ниво 2: карта 15 × 10, 3 чудовища, 2 съкровища.

всяко следващо ниво се определя като сума на показателите от предишните две:

ниво 3: карта 25 × 20, 5 чудовища, 4 съкровища.

ниво 4: карта 40 × 30, 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

Помислете върху следните въпроси и изберете подходящи допускания при реализация на прогамата си:

- 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?