Теми за проекти

*курс Обектно-ориентирано програмиране*

*за специалности Информатика и Информационни системи*

*летен семестър 2021/2022 г.*

# Обща информация за проектите

Проектът е цялостна задача, която трябва да решите с помощта на познанията по C++, получени през летния семестър. Правилата за завършване на курсовете по ООП и ООП-практикум с изпратените проекти са следните:

* Срок за предаване на **завършените** проекти: 24.06.2022 г.
* В обявения срок трябва да предадете:
  + Документация на проекта;
  + Изходен код на решението;
  + Няколко примера, подбрани от Вас, които демонстрират работата на задачата.
* Предаването става чрез прикачване на ZIP архив към съответното задание в Moodle, който съдържа всички файлове, необходими за компилирането на проекта, и документацията към проекта.
* Документацията на проекта трябва да съдържа:
  + Анализ на задачата и вашия подход за решение (на какви стъпки сте разделили решението, какъв метод или алгоритъм сте избрали, как сте се справили с конкретен проблем);
  + Кратко описание на класовете, създадени от Вас за решение на задачата (избраната архитектура, описание на член-данните и член-функциите на класовете и начинът им на използване);
  + Идеи за бъдещи подобрения.
  + Примерно съдържание на документацията е описано по-долу.
  + Препоръчваме на студентите да разучат някоя система за генериране на документация от коментари в изходния код, като [Doxygen](http://www.doxygen.nl/) или всяка друга аналогична система. При използване на такава система отпада изискването да се предава текстовата документация, описана по-горе, но се изисква описание в коментарите на кода на създадените класове, както и на техните методи и член данни. Като част от курсовия проект трябва да се предадат и генерираните файлове с документация.
* Всички проекти, които работят с дати, да ги представят във формата [ISO 8601](https://bg.wikipedia.org/wiki/ISO_8601).
* По време на защитата трябва да разкажете в рамките на 10 минути вашето решение и да демонстрирате работата на програмата с подготвени от вас данни.
* 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.
* Възможно е даването на малка задача за допълнение или промяна на функционалността на проекта ви, която вие трябва да реализирате за максимум 1 час.
* Невъзможност да реализирате малката задача на място означава, че не познавате добре проекта си и поражда съмнения, че сте ползвали чужда помощ за реализацията му. Последното ще се отрази негативно на оценката ви.
* Установено плагиатство на код от колеги и от други източници води до анулиране на работата и оценка Слаб 2 за курсовете по ООП и ООП-практикум. За плагиатство се счита използване на какъвто и да е чужд код в решението, чиито източник не е изрично упоменат в документация и като коментар в кода Ви.

### Критерии за оценка на проекта

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.

Не се позволява използване на функционалност и библиотеки извън тези, декларирани в следните заглавни файлове: *<iostream>*, *<fstream>, <new>, <cstring>, <cmath>, <exception>, <stdexcept>, <string> и <vector>.* Ако се искате да използвате определена библиотека, която не е включена в списъка, обърнете се към титулярите на курса за разрешение.

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.
* Дали е извършен анализ на поставената задача и на различните подходи за нейното решаване.
* Каква част от необходимата функционалност е била реализирана.
* Дали са обхванати всички потенциални сценарии за използване на функционалността.
* 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.
* Дали е постигната оптимална сложност на реализираните алгоритми (ако има такива и спрямо текущите ви познания).
* Дали е постигнато оптимално управление на паметта, включително заделяне и освобождаване на динамична памет, статични член-данни, глобални променливи.
* 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.
* Дали коректно са прихванати и обработени възможните грешки от различен характер - грешен или некоректен вход от потребителя, проблеми с външни ресурси (памет, файлове) и др. При възникнали проблеми програмата трябва да реагира адекватно и да се стреми да запази максимално данните коректни и цели. Да не се допуска изтичане на памет и всякакви други ресурси.
* Качество и пълнота на представената документация.
* Идеи и потенциал за бъдещи разширения.
* Представяне на проекта.
* Възможност за бърза реализация на малка промяна в проекта (познаване на кода) и отговаряне на въпроси.

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 използваме 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.

### Примерна структура на документацията

Съдържание(препоръчителен обем без приложенията: от 3 до 5 стр.)

**Глава 1. Увод** (1 стр.)

1.1. Описание и идея на проекта (3–4 изр.)

1.2. Цел и задачи на разработката (½–1 стр.)

1.3. Структура на документацията (3–4 изр.)

**Глава 2. Преглед на предметната област** (½–1 стр.)

2.1. Основни дефиниции, концепции и алгоритми, които ще бъдат използвани

2.2. Дефиниране на проблеми и сложност на поставената задача

2.3. Подходи, методи (евентуално модели и стандарти) за решаване на поставените проблемите

**Глава 3. Проектиране** (1–2 стр.)

3.1. Обща архитектура – ООП дизайн

3.2. Диаграми (на структура и поведение - по обекти, слоеве с най-важните извадки от кода)

**Глава 4. Реализация, тестване** (1–2 стр.)

4.1. Реализация на класове (включва важни моменти от реализацията на класовете и малки фрагменти от кода)

4.2. Управление на паметта и алгоритми. Оптимизации.

4.3. Планиране, описание и създаване на тестови сценарии (създаване на примери)

**Глава 5. Заключение** (2–3 изр.)

5.1. Обобщение на изпълнението на началните цели

5.2. Насоки за бъдещо развитие и усъвършенстване

**Използвана литература**

### Изисквания за оформяне на документацията на проекта:

1. Шаблонът е препоръчителен и може да се променя в зависимост от конкретното задание.
2. Йерархията на структуриране на съдържанието да не бъде повече от 3 нива, номерирани с арабски цифри – напр. 1.2.3.
3. Чуждестранните термини да бъдат преведени, а където това не е възможно – цитирани в *курсив* и нечленувани.
4. Страниците да бъдат номерирани с арабски цифри, в долния десен ъгъл.
5. Използваният шрифт за основния текст на описанието да бъде Times New Roman 12 или Arial 10, и Consolas за кода, с междуредие 16pt.
6. Да се избягва пренасянето на нова страница на заглавия на секции, фигури и таблици.
7. Да се избягват празни участъци на страници вследствие пренасянето на фигури на нова страница.
8. Всички фигури и таблици да бъдат номерирани и именовани (непосредствено след фигурата или таблицата).
9. Всички фигури и таблици да бъдат цитирани в текста.
10. Всеки термин, дефиниция, алгоритъм или информация, която е взета от литературен източник или Интернет трябва да бъде цитирана.
11. Всички цитати да бъдат отразени в списъка на използваната литература.
12. Всички източници от списъка на използваната литература да бъдат цитирани в текста.

Използваната литература да се цитира съгласно MLA Style —<http://www.library.mun.ca/guides/howto/mla.php>

# Съдържание

[**Обща информация за проектите**](#_trbb44c0k51g) **1**

[Критерии за оценка на проекта](#_vfwltab6lbm) 2

[Примерна структура на документацията](#_miyibgiljt2b) 4

[Изисквания за оформяне на документацията на проекта:](#_mn7y53o7g4tj) 5

[**Съдържание**](#_iwvh897cya6w) **6**

[**1: JSON Parser**](#_hbsj1abj60m9) **8**

[Извеждане на екрана](#_aiocw0d8c27d) 8

[Търсене по ключ](#_4eg0yxudbggw) 8

[Промяна на обект](#_12pucvusxq33) 8

[Създаване на обект](#_6rlb4lxslv2h) 8

[Изтриване на елемент](#_7xfanfvindar) 9

[Преместване](#_ibuz8a8ohupi) 9

[Итерация](#_z6qpjcov6g1n) 9

[**2: Приложение за работа с електронни таблици**](#_cb4fvppyzjfq) **10**

[Представяне на данните](#_7wztqf7bovl) 10

[Типове данни в таблицата](#_7z5osvq5ap6w) 10

[Зареждане на таблица](#_533weeal5c25) 11

[Записване на таблица](#_aoahug3663af) 12

[Извеждане на таблицата на екрана](#_q15wksdgig1q) 12

[Редактиране на клетки](#_oeevnvrhqd33) 12

[Сортиране](#_p0bjxz3msxv8) 13

[Създаване на извадка](#_luogf4fk0sla) 14

[**3: Приложение за работа с растерни изображения**](#_z3kkky42i5bx) **16**

[Създаване на потребителска сесия](#_b6l4l5ay2p62) 16

[Конвертиране до черно-бяло изображение](#_5unre7lo3eqt) 16

[Конвертиране до монохромно изображение](#_krmh0o3hvvrp) 16

[Негатив на изображение](#_qhgr99x3v78x) 17

[Завъртане на изображение](#_6jt6ylgfmz07) 17

[Обръщане на изображение](#_y806ioizegol) 17

[Отказ от последна транформация](#_vqvhtu4lcykq) 17

[Добавяне на изображение](#_6pqsdvvncac2) 17

[Запис на промените](#_iiwlwbothein) 17

[Информация за текуща сесия](#_gajgmqd5i6l8) 18

[Множество сесии](#_rxeds0csa3rw) 18

[**4: Приложение за работа с векторни изображения**](#_uw1pebaeex7w) **20**

[Отваряне на файл с изображение](#_m72uq3ta3os6) 20

[Извеждане на изображение](#_ohuncgadawci) 20

[Създаване на нова фигура](#_urtl8dff2gbk) 21

[Групиране на елементи](#_xbznt61wwjbm) 21

[Преместване на елементи в различни слоеве](#_s3sokljodog4) 21

[Транслиране на елементи](#_3g1vvnw9b6oc) 22

[Скалиране на елементи](#_mld7kmeab8z7) 22

[Завъртане на елементи](#_yk9o624duvsa) 22

[Записване на изображение](#_cvozfknf8ef7) 23

[Извеждане на познатите команди](#_djqf6ng7n209) 23

[**5: Поддържане на HTML**](#_1bhm3lkasro0) **24**

[Зареждане на HTML файл](#_r1nykwu7f48r) 24

[Създаване на нов HTML файл](#_lzape4u653nw) 24

[Записване на файл](#_w45y6u2brefv) 24

[Избор на елементи](#_iwj03ooffj48) 24

[Промяна на текст на елемент](#_y7ogo29ta6g0) 25

[Промяна на текст на атрибут](#_p6gkjg1gi50h) 25

[Добавяне на нов елемент](#_9b5dcou61m8d) 25

[Премахване на елемент](#_2sz4rwx073zr) 25

[Преместване на елемент](#_bno18vb313qn) 26

[**6: XML Parser**](#_oe48n372oz3) **27**

[Извеждане на екрана](#_3gpcbs13ugzv) 27

[Задаване на текущ елемент](#_jqkxmjbhkq4f) 27

[Задаване на стойност на атрибут](#_akc8w3g3s9sv) 28

[Достъп до вложени елементи](#_6mfe5vf6q7tn) 28

[Достъп до текста на елемент](#_zfmkgj3frkp7) 28

[Задаване на текст на елемент](#_6lmu6tz6vm7) 28

[Изтриване на атрибут](#_s3vcfoxv8n3a) 28

[Добавяне на нов наследник](#_j6ttp03dsb3n) 28

[Прости заявки](#_sph2j5rv8nnx) 29

[**7: Библиотека**](#_39gl91ga1ni6) **30**

[Вход в системата](#_cu07r48z7nbb) 31

[Изход от системата](#_l8hvjnxh6mwe) 31

[Команди за извеждане на записи](#_h93wwxls7oim) 31

[Команди за търсене на записи](#_jqj5qz8a0cgq) 31

[Търсене на потребители](#_4jpo9yc10ngu) 32

[Добавяне на потребител](#_mcw5f7c6hg3h) 32

[Премахване на потребител](#_b8a09zhmm9am) 32

[Промяна на паролата](#_6f22n8dbr2mr) 33

[Заемане на книга](#_92wxq6oqb83a) 33

[Връщане на книга](#_qnmyx1t3glb2) 33

# 

# 1: JSON Parser

[JSON](https://www.json.org/json-bg.html) е популярен текстов формат за описване на данни. Вашата задача е да направите програма, която работи с файлове в такъв формат.

При подаване на даден файл, вашата програма трябва да го прочете и да провери дали той е валиден спрямо синтаксиса на JSON. Ако има някакъв проблем, тя трябва да съобщи максимално ясно какво и къде не е наред, така че потребителят да може да го поправи.

Ако всичко с файла е наред, вашата програма трябва да го прочете и съхрани в обект от подходящо реализиран от вас клас.

Програмата ви трябва да поддържа следните команди.

### Извеждане на екрана

> **print**

При подаване на подходяща команда програмата трябва да изведе на екрана съдържанието на обекта. Постарайте се да е четивно.

### Търсене по ключ

> **search <key>**

При подаване на подходяща команда, следвана от стрингов ключ <key>, вашата програма трябва да провери дали в обекта се съдържат данни, записани под този ключ, и ако да, да създаде масив от всички такива данни. Всяка от тях да бъде представена като JSON обект. При по-късно поискване, този масив трябва да позволява извеждане на екрана или запис във файл, както на целия масив, така и на отделен негов елемент по индекс. Реализирайте оператор за индексиране на вашия обект за тази цел.

### Промяна на обект

> **edit <element> <value>**

При подаване на пълен път към даден елемент <element> и стрингова стойност <value>, вашата програма трябва да замени стойността на посочения елемент с парснатата, според синтаксиса на JSON, стойност на низа, ако такъв елемент съществува и ако символният низ е коректен. В противен случай, трябва да обяви каква е грешката.

### Създаване на обект

> **create <element> <value>**

При подаване на пълен път към даден елемент <element> и стрингова стойност <value>, вашата програма трябва да добави такъв елемент и да му присвои парснатата, според синтаксиса на JSON, стойност на низа. Ако такъв елемент съществува, това трябва да се обяви като проблем. Иначе, да се добавя, евентуално с рекурсивно създаване на целия път. Ако символният низ или пътят не са коректни, да се съобщи с подходящо описание на грешката.

### Изтриване на елемент

> **remove <element>**

При подаване на пълен път до елемент <element>, той да се изтрие. Ако такъв елемент не съществува, да се изведе грешка.

### Преместване

> **move <from> <to>**

При подаване на два пътя, всички елементи, намиращи се на първия път <from>, да бъдат преместени върху втория <to>.

### Итерация

Вашият клас трябва да поддържа концепцията за итератор. Чрез него трябва да може да се мине през всички елементи на даден JSON обект и те да могат да се прочетат. Желателно е да има възможност за редакция на даден елемент, без промяна на типа му. Бонус би била пълна възможност за редакция през итератор.

Когато записвате данните във файл, трябва да може да се укаже как да се подравни информацията. Трябва да се поддържат поне две опции:

* подравнено с отмествания и нови редове, с цел по-добра четивност;
* с минимален обем, без излишни празни позиции.

Освен това, програмата ви трябва да може да запише в него или целия документ, или отделна негова част така, че да се получи валиден JSON формат.

За целта или разширете командите save/saveas така, че те да приемат допълнителни аргументи, или добавете нови команди.

**По-конкретно за записването на извадка**

При подаден път или ключ, във файла се записва елементът, намиращ се на дадения път (ако такъв съществува) или масив от всички елементи с даден ключ. Ако съответното съдържание не съществува, да се съобщи на потребителя.

# 2: 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 '-'.

Пример:

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 '-'.

Пример:

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 \\.

Пример:

"Hello world!"

"C:\\temp\\"

"\"This is a quotation\""

**Валута** – число с плаваща запетая, форматирано с точно два символа след десетичната запетая и символ за валута. Поддържат се три валути: левове, евро и долари.

Пример:

$ 2.12

€ 15.30

2018.05 лв.

### 

След като вашето приложение отвори даден файл, то трябва да може да извършва посочените по-долу операции.

### Зареждане на таблица

> **load <file>**

Зарежда таблица от подаден файл. Файлът може да е указан с пълен път или да бъде в директория по подразбиране.

Ако при зареждането на данните, приложението ви открие грешка, то трябва да изведе подходящо съобщение 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 кои символи 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”*.

### Записване на таблица

> **save <file>**

Таблицата се съхранява в посочения файл.

### Извеждане на таблицата на екрана

> **print**

При извеждане на заредената таблица, 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

> **edit <parameters>**

Командата edit трябва да позволява (с подходящи параметри) на потребителя да променя стойностите на отделните клетки. Това става като се укажат редът и колоната на клетката, която ще бъде променена, а също и каква стойност да се запише в нея. Потребителят може да въведе произволен тип данни, който се поддържа от вашата програма (например цяло число, дробно число, низ и т.н.). Забележете, че по този начин може да се промени типът на дадена клетка, например от число, тя може да стане символен низ.

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.

### Сортиране

> **sort <column> [<asc>|<desc>]**

При поискване на подреждане на редовете, трябва да се подаде номер на колона <column>, според която да се подредят данните. Номерата на колоните започват от 1. Може да се подаде и параметър, указващ дали подреждането да е в нарастващ <asc> или намаляващ ред <desc>.

При сравняване на два елемента от един и същ тип се прилага естествената за типа наредба, с изключение на валутата. Валути се сравняват като бъдат превалутирани до български лева.

При сравняване на данни от различен тип се прилага следното правило:

Числата се сравняват естествено.

При сравняване на валута с който и да е друг тип се използва числовата стойност на валутата, преобразувана в лева.

При сравняване на низ с число, ако низът може да се преобразува към число (т.е. е валидно число) се работи с тази стойност. Ако не е валидно число, се приема стойност 0.

Следва примерна таблица и няколко начина, по които се сортира.

| 1 | $ 100.12 | text1 |
| --- | --- | --- |
| 2.123 | 150.02 лв. | baba |
| some data | Text data | 12.4 |
| text | 432 | 123 |

Подредба в нарастващ ред по колона 1

| some data | Text data | 12.4 |
| --- | --- | --- |
| text | 432 | 123 |
| 1 | $ 100.12 | text1 |
| 2.123 | 150.02 лв. | baba |

Подредба в нарастващ ред по колона 2

| some data | Text data | 12.4 |
| --- | --- | --- |
| 2.123 | 150.02 лв. | baba |
| 1 | $ 100.12 | 123 |
| text | 432 | text1 |

Подредба в нарастващ ред по колона 3

| 2.123 | 150.02 лв. | baba |
| --- | --- | --- |
| 1 | $ 100.12 | text1 |
| some data | Text data | 12.4 |
| text | 432 | 123 |

### Създаване на извадка

Вашата таблица трябва да поддържа функционалност за извличане на под-таблица. Това може да стане по няколко начина.

* Чрез подаване на диапазон - номер на начални и крайни ред и колона (включително). Както редовете, така и колоните номерираме последователно, започвайки от 1.

Например чрез заявката 1:1 - 10:14 трябва да се извлече частта от таблицата, започваща на първи ред, първа колона и завършваща на десети ред и четиринадесета колона - реално десет реда и четиринадесет колони.

* Чрез подаване на списък от колони, които ни интересуват. Извличат се всички редове. Номерата на колоните не е задължително да са изброени в нарастващ ред, но в извадката се запазва реда от оригиналната таблица.

Например чрез заявката {1, 3, 4, 2, 6} се извличат първите четири и шестата колона.

* Чрез подаване на критерий за колона - извличат се всички колони, но само редовете, за които даденият критерий е валиден. Валидни критерии са операторите за сравнение и равенство (<, >, <=, >=, ==, !=). Не се изисква възможност за подаване на сложни заявки (с логически операции), но реализация на тази функционалност ще бъде третирана като бонус.

Например чрез заявката #3 < 10 трябва да се извлекат всички редове, за които стойността в третата колона е по-малка от 10.

Имате свобода да променяте или дефинирате синтаксиса за определяне на критериите за извадка. Ваша отговорност е да се погрижите да откриете потенциални проблеми или грешки в заявката и да изведете подходящо съобщение на потребителя.

# 3: Приложение за работа с растерни изображения

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>). Използвайки библиотека, включително написана от вас, можете да поддържате лесно и BMP изображения.

### Създаване на потребителска сесия

При стартиране на вашето приложение, то първоначално не разполага със заредени файлове за обработка, както и текущи сесии. За да се заредят файлове във вашата програма трябва да се стартира нова сесия. Това действие се извършва чрез командата **load*,*** като след нея се въвеждат пътища към файловете, които да бъдат заредени в сесията. Тази команда генерира и пореден номер на сесия, по който по-късно тя може да бъде идентифицирана, в случая на множество потребителски сесии.

Пример:

**> load** image01.ppm image02.pgm

*Session with ID: 1 started*

Стартира сесия, в която ще се обработват изображенията image01.ppm и image02.pgm. Зареждайки файловете, трябва да можете да изпълнявате различни трансформации върху тях. Когато се прилагат трансформации в сесия, то те важат за всички заредени изображения за **текущата** сесия.

### Конвертиране до черно-бяло изображение

**> 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 черни, бели или нюанси на сивото).

### Конвертиране до монохромно изображение

**> monochrome**

Поведението на тази операция е като това на предишната, но тук файлът се конвертира до монохромен (такъв, в който има само черни и бели пиксели, без никакви нюанси на сивото). Отново, ако входният файл е монохромен, програмата не прави нищо.

### Негатив на изображение

**> negative**

Тази команда прави негатив (тонално обръщане) на изображенията в текущата сесия.

### Завъртане на изображение

**> rotate left**

или

**> rotate right**

Тези команди реализират завъртане на изображение на ляво и дясно респективно на 90 градуса. Обърнете внимание, че четири приложения на ротация в една посока всъщност е идентитет и следователно няма нужда да бъдат прилагани всичките четири ротации.

### Обръщане на изображение

**> flip horizontal**

или

**> flip vertical**

Тези команди реализират обръщане на изображението хоризонтално или вертикално. Обърнете внимание, че две последователни прилагания на обръщане в посока всъщност е идентитет.

### Отказ от последна транформация

**> undo**

Като един истински редактор за изображения, вашето приложение трябва да поддържа и команда **undo.** Тя премахва последно направената транформация в текущата сесия. Ако е стартирана нова сесия и след това веднага бъде въведена команда **undo**, то тя не трябва да има никакъв ефект.

### Добавяне на изображение

**> add** image02.ppm

Добавя ново изображение към текущата сесия. Всички приложени до момента трансформации не се прилагат върху него.

### Запис на промените

**> save**

За запис на направените трансформации е необходимо да реализирате команда

**save**. Тя записва новосъздадените файлове в текущата директория, в която се намират и оригиналните изображения в сесията, като използва същото име, но с добавено времето, когато е записан файла.

Пример:

**> load**image01.ppm

Session with ID: 1 started

**> grayscale**

**> save**

Записва нов файл с име image01\_<modified\_time>.ppm в директория, в която се намира image01.ppm.

### Информация за текуща сесия

**> session info**

Имплементирайте функционалност, която да дава възможност на потребителя да получава подробна информация за текущата потребителска сесия - нейният идентификационен номер, участващите изображения, както и набора от трансформации, които са приложени до този момент върху изображенията, участващи в сесията.

Пример**:**

**> load** image1.ppm image2.pgm image3.pbm

Session with ID: 1 started

**> rotate left**

**> grayscale**

**> session info**

Name of images in the session: img1.ppm img2.pgm

Transformations: rotate left, grayscale

### Множество сесии

Имплементирайте поддържането на множество потребителски сесии (като минимум поне 2), както и възможност за превключване между всяка една от тях посредством нейният идентификационен номер (ако сесия с такъв номер съществува, иначе трябва удачно да се извести на потребителя, че такава сесия не съществува).

Пример:

**> load** img1.ppm img2.pgm

Session with ID: 1 started

**> grayscale**

**> rotate left**

**> load** img3.pbm img4.ppm

Session with ID: 2 started

**> rotate right**

**> switch 1**

You switched to a session with ID: 1!

Name of images in the session: img1.ppm img2.pgm

Transformations: grayscale, rotate left

**> switch 2**

You switched to a session with ID: 2!

Name of images in the session: img3.pbm img4.ppm

Transformations: rotate right

**> exit**

# 

# 4: Приложение за работа с векторни изображения

В рамките на този проект трябва да се реализира приложение, което може да нарежда фигури (елементи) във векторни изображения. Фигурите могат да се припокриват, т.е. могат да бъдат в различни слоеве на изображението. Приложението трябва да може да зарежда фигури от файл, да извършва върху тях дадени операции, след което да може да записва промените обратно на диска.

Поддържат се четири основни вида фигури: отсечки, окръжности, правоъгълници и многоъгълници. Дизайнът на приложението обаче трябва да е такъв, че да позволява при нужда лесно да може да се добави поддръжка на нови фигури. За всяка фигура се задава цвят на контурната линия, както и цвят на запълване (или прозрачно).

Отсечките се определят от координатите на двата си края, окръжностите - с координати на центъра и радиуса им, правоъгълниците - с координати на горен-ляв ъгъл, ширина и дължина, а многоъгълниците - с координати на последователните си върхове.

Потребителят трябва да може да изпълнява следните команди.

### Отваряне на файл с изображение

> **open** <файл>

Прочита съдържанието на даден файл и го зарежда в паметта. Когато се зарежда съдържанието на даден файл, трябва да се прочетат само фигурите, които се разпознават от приложението. Всички останали елементи на файла се игнорират, но се извежда подходяща информация за потребителя.

Файлът може да се зададе с пълния си път, в противен случай се търси в папка по подразбиране.

Пример:

> open figures.svg

Successfully opened figures.svg

### Извеждане на изображение

> **print**

Извежда на екрана фигурите, прочетени от файла.

Пример:

> **print**

1. rectangle 5 5 10 10 green orange
2. circle 5 5 10 blue green
3. rectangle 100 60 10 10 red white

### Създаване на нова фигура

> **create** <тип на фигура> <компоненти на фигурата>

Създава нова фигура и я добавя към изображението.

<тип на фигура> е някой от типовете фигури, които се разпознават от приложението.

<компоненти на фигурата> са всички необходими за инициализиране на фигурата компоненти. Всяка фигура получава уникален идентификатор при създаването си.

Пример:

> create rectangle 100 100 10 20 yellow white

Successfully created rectangle. ID 4.

### Групиране на елементи

> **group** <region>

<region> се задава от потребителя като координати на горен-ляв и долен-десен ъгъл.

Пример:

> group 10 10 15 15

Командата групира в нов елемент всички фигури, които изцяло се съдържат в указания регион. Резултатът е нова фигура, като всички операции, прилагани върху нея, се случват върху елементите в нея. Върху елементи, част от група, не могат да се прилагат индивидуални операции.

> **ungroup** <id>

Прекратява групирането на елементите в посочената с <id> група. Групата вече не съществува (няма фигура с такъв идентификатор), а фигурите, съставляващи групата, могат да се променят индивидуално.

### Преместване на елементи в различни слоеве

> **bring forward** <id> <n>

> **send backward** <id> <n>

Всеки елемент може да се мести напред - назад в стека (едно ниво или най-отгоре/най-отдолу). <id> е идентификаторът на елемента в изображението. <n> е броят нива, с които да се премести. Номерирането на нивата започва от най-долу.

Пример:

> bring forward 2 2

Ако фигурите са изведените с командата print, то командата bring forward ще премести окръжността най-отгоре.

### Транслиране на елементи

> **translate** [<id>] vertical=<v> horizontal=<h>

Ако е посочен номер на елемент в изображението, то избраният елемент се транслира с указания вектор. Ако не е посочен номер на елемент, то операцията се прилага за всички елементи в изображението.

Пример:

> translate 2 vertical=10 horizontal=100

> print

1. rectangle 5 5 10 10 green orange
2. rectangle 110 160 10 10 red white
3. circle 5 5 10 blue green

Транслира се единствено правоъгълникът, който се намира във втория слой на изображението.

### Скалиране на елементи

> **scale** [<id>] <коефициент за ширината> <коефициент за ширината>

Скалира посочен елемент (или всички) с указаните коефициенти за скалиране.

Пример:

> scale 1 2 3

> print

1. rectangle 5 5 20 30 green orange
2. rectangle 110 160 10 10 red white
3. circle 5 5 10 blue green

### Завъртане на елементи

> **rotate** [<id>] <ъгъл в градуси>

Завърта посочен елемент (или всички) на указания ъгъл. Център на ротацията е центърът на ограждащия елемента (елементите) правоъгълник.

### Записване на изображение

> save

> save as <форматът, който се поддържа от вашето приложение>

Приложението трябва да поддържа два вида извеждане: в избран от вас формат и в някакъв графичен формат, който може да се визуализира (ppn, bmp, svg или друг).

### Извеждане на познатите команди

> help

Извежда списък на командите, които се поддържат от приложението.

# 

# 5: Поддържане на HTML

Да се напише програма, реализираща четене и операции върху HTML файлове.

Да се поддържат уникални идентификатори на елементите 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, то само първото срещане се отчита, а всички останали се считат без идентификатор.

При стартиране на приложението, потребителят може да зареди съществуващ HTML файл или да създаване на нов (само с глава и тяло). Тези операции са възможни със следните команди:

### Зареждане на HTML файл

> load <html\_file>

Зарежда HTML файл <html\_file>, зададен с пълния път до файла или в директория по подразбиране, ако е посочено само името на файла.

### Създаване на нов HTML файл

> new

Създава нов HTML файл, който съдържа само глава и тяло. Името на новия файл е new.html. Ако такъв файл съществува, да се добави номер, така че да стане уникален (например, ако файл new.html и new01.html вече съществуват, името да е new02.html).

Следващите команди, които трябва да поддържа вашето приложение, могат да бъдат използвани само, ако вече има зареден файл.

### Записване на файл

> save

Записва текущия файл.

> save as <new\_name>

Записва текущата версия на файла под ново име, посочено като аргумент на командата.

### Избор на елементи

> select id <value>

Извежда елемента с посочения идентификатор, ако такъв съществува. Ако такъв елемент не съществува, се извежда подходящо съобщение.

> select\_list tag <tag\_value> [attribute <attribute\_value>]

Извежда всички елементи с посочения таг. Ако в командата е посочен и атрибут, от всички елементи с посочения таг се филтрират само онези, които притежават и указания атрибут.

> select\_one tag <tag\_value> [attribute <attribute\_value>] <index>

След селекция на всички елементи с посочения таг и атрибут, се избира един от елементите, намиращ се на съответния индекс. Този елемент става текущ и може да се използва в следващите команди, при липса на подаден идентификатор.

### Промяна на текст на елемент

> **set <value> [<id>]**

Променя текста на елемента с дадения идентификатор или текущия.

### Промяна на текст на атрибут

> **set attribute <key> <value> [<id>]**

Променя текста на атрибута с ключ <key> на елемента с дадения идентификатор или текущия. Това е начин да се зададе идентификатор на елемент без такъв. Да не се позволява съществуване на елементи с дублиращи идентификатори!

### Добавяне на нов елемент

> **add <tag> [<id>]**

Добавя нов елемент с таг посочения <tag> като дете на елемента с идентификатор <id> или на текущия.

### Премахване на елемент

> **delete [<id>] [sc]**

Изтриване на елемента с посочения идентификатор или текущия. Ако в него има вложени елементи, те могат да бъдат запазени, ако самият той е вложен в друг елемент и е избрана опцията sc (save children). В този случай децата се прехвърлят към родителския елемент. Ако опцията не е избрана или елементът с посочения идентификатор не е вложен, децата също трябва да бъдат премахнати.

### Преместване на елемент

> **move [from <id>] [to <id>]**

Елементът с посочения за from идентификатор <id> се премества като вложен в елемента с to идентификатор <id>. Ако единия от тези идентификатори (заедно с определящата дума from/to) липсва, се взима текущият. Не се позволява и двата да липсват!

# 6: XML Parser

Да се напише програма, реализираща четене и операции с [XML](http://en.wikipedia.org/wiki/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 по някакъв начин от програмата.

След като приложението отвори даден файл, то трябва да може да извършва посочените по-долу операции, както и стандартните операции open, close, save, save as, help и exit.

### Извеждане на екрана

> **print**

Извежда на екрана прочетената информация от XML файла (в рамките на посочените по-горе ограничения за поддържаната информация). Извеждането трябва да е XML коректно и да е “красиво”, т.е. да е форматирано визуално по подходящ начин (например, подчинените елементи да са по-навътре).

> **show**

Извежда на екрана прочетената информация от XML файла (в рамките на посочените по-горе ограничения за поддържаната информация). Извеждането трябва да е XML коректно и да е с минимален брой символи, без да се променя семантиката.

### Задаване на текущ елемент

> **select <id>**

Задава за текущ елемент този с подадения идентификатор. Ако няма такъв елемент, да се реагира по подходящ начин и да няма текущ елемент. Текущ елемент се използва, когато не е зададен идентификатор в някоя от следващите команди.

### Задаване на стойност на атрибут

> **set <key> <value> [<id>]**

Присвояване на стойност на атрибут с ключ <key> на елемента с дадения идентификатор. Ако идентификатор не е зададен, да се ползва текущия елемент.

### Достъп до вложени елементи

> **children [<id>]**

Връща списък на вложените елементи на елемента с посочения идентификатор. Ако идентификатор не е зададен, да се ползва текущия елемент.

> **child <n> [<id>]**

Достъп до n-тия наследник на елемента с посочения идентификатор. Ако идентификатор не е зададен, да се ползва текущия елемент.

### Достъп до текста на елемент

> **text [<id>]**

Достъп до текста на елемента с посочения идентификатор. Ако идентификатор не е зададен, да се ползва текущия елемент.

### Задаване на текст на елемент

> **set text [<id>] “<text>”**

Задаване на нов текст на елемента. Подаденият текст се подава в кавички. Може да съдържа множество нови редове.

### Изтриване на атрибут

> **delete <key> [<id>]**

Изтриване на атрибут на елемент по ключ. Ако идентификатор не е зададен, да се ползва текущия елемент.

### Добавяне на нов наследник

> **newchild [<id>]**

Добавяне на НОВ наследник на елемент. Новият елемент няма текст и никакви атрибути, освен идентификатор. По-късно може да бъде променян.

### Прости заявки

> **xpath <XPath> [<id>]**

Oперации за изпълнение на прости [XPath 2.0](http://en.wikipedia.org/wiki/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 поддържа оператор / (например “person/address” дава списък с всички адреси във файла).
* Да поддържа оператор [] (например “person/address[0]” два адресът на първия елемент във файла).
* Да поддържа оператор @ (например “person(@id)” дава списък с id на всички елементи във файла).
* Да се поддържат 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 ООП и 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](http://www.w3.org/TR/xml-names/).
* Да се реализират различните XPath оси (ancestor, child, parent, descendant,...).

# 7: Библиотека

Да се реализира информационна система, която поддържа библиотека. В библиотеката се съхраняват и предоставят на читателите както книги, така и периодични издания (вестници, списания). Програмата ви трябва да съхранява и обработва данни за наличните в момента книги и издания, както и за читателите в един или повече файлове. Форматът на представяне на данните е по ваш избор. Опишете го в документацията на проекта.

Всяка книга се характеризира със следните данни:

* автор;
* заглавие;
* издателство;
* жанр;
* кратко описание;
* година на издаване;
* ключови думи;
* рейтинг;
* уникален номер за библиотеката;
* незадължителен ISBN номер.

Всяко печатно издание има:

* заглавие;
* издателство;
* жанр;
* кратко описание;
* година и месец на издаване;
* брой;
* рейтинг;
* уникален за библиотеката номер;
* незадължителен ISSN номер;
* списък от статии (съдържание) - всяка от тях има заглавие, автор и ключови думи.

Системата поддържа също два вида потребители - администратори на библиотеката и читатели. Всеки потребител се характеризира със следните данни:

* потребителско име;
* парола;
* дата на регистрация;
* дата на последен достъп до системата.

За читателите трябва да се пази списък от книги или издания, които са заемали - всяко с дата на заемане и дата на връщане (реално връщане, ако книгата е върната или срок на връщане, ако все още е заета).

За администраторите - служебен имейл адрес и отдел, за който отговарят.

При стартиране на приложението се стартира данните се зареждат от файловете. Системата трябва да поддържа следните команди.

### Вход в системата

> 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.”

### Изход от системата

> logout

Потребителят напуска системата (програмата продължава да работи).

### Команди за извеждане на записи

> book all

Извежда последователно за всички книги в библиотеката следната информация: заглавие, тип, жанр, персонален номер.

> series all

Извежда последователно за всички печатни издания в библиотеката следната информация: заглавие, тип, жанр, персонален номер.

> list all

Извежда последователно за всички книги и печатни издания в библиотеката следната информация: заглавие, тип, жанр, персонален номер.

### Команди за търсене на записи

> list info <isbn\_value>

Извежда на екрана подробна информация за изданието с ISBN/ISSN номер равен на <isbn\_value>.

> books find <option> <option\_string> [sort <key> [asc|desc] ]  
> series find <option> <option\_string> [sort <key> [asc|desc] ]  
> list find <option> <option\_string> [sort <key> [asc|desc] ]

<option>е едно от title, author, tag (ключова дума).

<option\_string>е стойността на критерия за търсене, може да съдържа интервали.

За периодичните издания да се проверяват авторите на всяка статия.

Ако е зададена опцията sort, то резултатът трябва да се изведе подреден по полето <key>. Опцията asc или desc определя дали да се подредят в нарастващ или намаляващ ред. Ако не е зададена, подредбата е в нарастващ ред.

Пример:

books find title Introduction to programming

books find author Stephen King

books find tag superhero sort year asc

### Търсене на потребители

> user find <option> <option\_string>

Изготвя списък на потребителите по даден критерий, зададен от <option> и <option\_string>. <option> е едно от name, ID, state. При опция name се извеждат данни за потребител с посоченото име. При задаване на ID се търси потребител, който е заел изданието със съответния идентификатор. state очаква следващият параметър да е едно от overdue, reader или inactive. overdue търси всички потребители, които имат книги, които не са върнати в срок. reader търси всички потребители, които са заемали повече от 5 книги в последния месец. inactive търси потребителите, които не са заемали книга повече от 3 месеца.

### Добавяне на потребител

> user add <user> <password> [admin]

Добавя нов потребител с потребителско име <user> и парола <password>. Потребителят и неговата парола се записват във файл. Ако е зададена опцията admin, новият потребител е администратор.

### Премахване на потребител

> user remove

Изтрива потребителя с потребителско име <user> от системата.

### Промяна на паролата

> user change [<name>]

Позволява промяна на паролата на потребител. Ако е зададено име на потребителя, се променя паролата на този потребител. В този форма командата изисква изпълняващият я да е администратор. Ако не е зададено име, се променя паролата на текущия потребител.

### Заемане на книга

> take <ID>

Изпълнява се само при вписан потребител. Заема книга/издание с определен вътрешен идентификатор. Съответната книга се маркира като заета и се вписва в записа на читателя.

### Връщане на книга

> return <ID>

Изпълнява се само при вписан потребител. Връща вече заета книга/издание с определен вътрешен идентификатор. Книгата вече се отбелязва като свободна.

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 ѝ 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? |
| --- | --- | --- |
| help | не | не |
| login | не | не |
| logout | да | не |
| exit | не | не |
| books all | да | не |
| books find | да | не |
| books view | да | не |
| books add | да | да |
| books remove | да | да |
| take / return | да | не |
| users add | да | да |
| users remove | да | да |
| user find | да | да |

* При обработване на командите да се вземат предвид потенциални грешки на потребителя (некоректно въведени команди или критерии) и да се извеждат подходящи помощни съобщения.
* Да се вземат предвид всякакви гранични случаи (например при добавяне на потребител вече съществува потребител с даденото име; опит да се върне книга, която не е заета или не е заета от този потребител).
* При добавяне на книга да се намери начин за генериране на уникални вътрешни идентификатори.
* Да се предвиди възможност не всяко издание да има всички характеристики.
* Да се отчетат всички проблеми при работа с файловете - проблеми с достъп до файла, както и потенциално повредени файлове.