Теми за проекти

*курс Обектно-ориентирано програмиране*

*за специалност Информатика летен семестър 2023/2024 г.*

***Последна редакция 21.03.2024***

21.03 - Проект 4. Променено име на команда за зареждане (open -> load)

# 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: **10.06.2024 г. (следете за промени)**
* В обявения срок трябва да предадете:
  + документация на проекта;
  + изходен код на решението;
  + няколко подбрани примера, които демонстрират работата на задачата.
* Предаването става чрез прикачване на ZIP архив към съответното задание в Moodle. Архивът трябва да съдържа файловете, 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, които създадени 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 разучаването на 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. Изисква се обаче описание в коментарите на кода на създадените класове, както и на техните методи и член-данни. Генерираните файлове с документация трябва да се предадат като част от курсовия проект.
* Всички проекти, които работят с дати, да използват формата [ISO 8601](https://bg.wikipedia.org/wiki/ISO_8601).
* По време на защитата в рамките на 10 минути трябва да разкажете за 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.
* По време на защитата може да ви бъде възложена малка задача за допълнение или промяна на функционалност на проекта ви, която трябва да реализирате за максимум 1 час.
* Невъзможност да реализирате тази задача на място означава, че не познавате добре проекта си и поражда съмнения, че сте използвали чужда помощ за реализацията му. Последното ще се отрази негативно на оценката ви.
* Установено плагиатство на код от колеги или 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.

### Критерии за оценка на проекта

Проектите се оценяват по редица 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. Решение, 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 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 (например готова библиотека), 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 е използван чужд код, сам по себе си той не носи точки. Допълнителни точки могат да се дадат или отнемат, според (1) способността ви за внедряване на код във вашето решение (например 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>

# Общи команди

Дадените по-долу команди се поддържат в голяма част 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 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 се избере подходящо представяне на информацията от файла.

Ако при зареждането на данните, приложението открие грешка, то трябва да се изведе подходящо съобщение за грешка и да се прекрати изпълнението.

> open C:\Temp\figures.svg

Successfully opened figures.svg

### Save

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

> save

Successfully saved figures.svg

### 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

### Exit

Излиза от програмата. Ако в този момент има някакви промени, които потребителят е правил, програмата трябва да го попита дали иска да ги запише преди излизане.

# Тема №1 Поредов текстов процесор

В рамките на този проект трябва да се реализира приложение, което зарежда текст като колекция от редове и може да прилага множество операции върху тях. Например търсене и замяна на текст, преобразуване (toUpper, toLower, …), премахване на водещи / завършващи празни позиции и т.н.

Има и операции работещи с колекцията редове като сортиране, разбъркване, премахване на повторения.

Някои редове са специални и работата с тях трябва да отчита това.

* Ако даден ред започва с число и точка, то той не трябва да се премества при операции сортиране.
* Ако ред е ограден с кавички (допускат се водещи или завършващи празни символи извън тях), то над него не се прилагат операции за премахване на празни позиции в кавичките, както и преобразуване към малки / главни букви.
* Редове, представляващи числа, се сравняват не лексикографски, а по числената си стойност.

Трябва да може да се дефинират блокове от редове и да се правят операции само в рамките на блок.

Трябва да се поддържа едновременно зареждане на множество файлове и превключване между активния в момента файл по неговото име.

Трябва да се поддържа отмяна на една или повече приложени операции (команда **undo**).

Трябва да се поддържат различни начини за извеждане на екрана (директно, wrap на определен брой колони, центрирано…).

Трябва да има възможност серия операции да се записват като макрос с име и да се прилагат по-късно по това име.

Разбира се, трябва да има възможност за редакция на редовете - промяна на съдържанието на даден ред, добавяне и премахване на редове.

Трябва да се поддържат команди **save** и **save as**, записващи текущото състояние на активния файл в оригиналното му име или под ново такова.

Да се поддържат и команди, даващи информация за текущо отворените файлове - за всеки неговото име, брой редове, брой символи и дали съдържа незаписани промени. При затваряне на текущия файл (команда **close**) или приключване на работа на програмата (команда **exit**) за всеки файл, съдържащ незаписани данни да се пита потребителя дали да се съхрани или те да се игнорират.

# Тема №2 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, дъги от 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 (където това има смисъл).

Отсечките се определят от координатите на двата си края; дъгите - с координати на центъра, радиуса си, ориентиран ъгъл на началото на дъгата и ширина на дъгата (и двете в градуси); правоъгълниците - с координати на горен-ляв ъгъл, ширина и дължина; а многоъгълниците - с последователните координатите на всичките си върхове. Обърнете внимание, че дъга с ширина по-голяма или равна на 360 градуса е окръжност.

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. arc 5 5 10 20 360 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 параметри. 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.

> **insert** <gr\_id> <elem\_id>

Добавя елемента с <elem\_id> към групата с <gr\_id>. Добавеният елемент сам по себе си може да е група.

> **extract** <gr\_id> <elem\_id>

Премахва елемента с <elem\_id> от групата с <gr\_id>.

### Промяна на видимост

> **hide** <id>

Скрива елемента <id>. Върху него продължават да се прилагат всички трансформации, но не се записва при save и не се извежда при print.

> **show** <id>

Отменя действието на извършено скриване на елемента <id>.

### 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. arc 5 5 10 20 360 black green

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

### Скалиране на елементи

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

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. arc 5 5 10 20 360 black green

### 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 (ppm, bmp, svg или друг).

### Извеждане на познатите команди

> help

Извежда списък на командите, които се поддържат от приложението.

# 

# Тема №3 Работа със CSV файлове

В рамките на този проект трябва да реализирате приложение, което зарежда таблици от CSV файлове и извършва серия операции над тях.

Таблиците могат да имат опционален първи ред с имена на колоните. Типът на данните в колоните трябва да се разпознае. **Всички елементи в една колона са от един и същ тип!** Валидни типове са текст, число, дата и факултетен номер от ФМИ. Трябва да се разпознават както класически факултетни номера от типа на 45123, така и новите от вида 0MI0400123. Не се предполага да валидирате контролната цифра в новите факултетни номера, но съблюдавайте валидни кодове на специалности. Ако данните в една колона са оградени в единични или двойни кавички, то тя винаги е от тип текст. Ако не можете да разпознаете типа на данните, отново третирате колоната като текстова.

Програмата може да зарежда данни от CSV файл (команда **open**) и да записва текущото състояние на данните (команди **save**, **save as**). При напускане на приложението (команда **exit**), ако има незаписани данни, потребителят трябва да потвърди дали да ги запази или да ги игнорира. Текущите данни могат да се извеждат на стандартния изход в подходящо форматиран вид (командата **print**).

Върху данните, заредени като таблица от CSV файл, са възможни следните операции:

* сортиране по номер или име на колона и посока (в нарастващ или намаляващ ред). За всички данни редът на сортиране е естественият за типа им, като за факултетните номера данните трябва да бъдат групирани по специалност, а в рамките на една специалност да бъдат подредени според реда на записване - първо старите факултетни номера в съответния ред, след това новите в същия ред;
* филтриране по номер или име на колона и условие - може да бъде ==, !=, <, <=, >, >= на подадена стойност (в съответния за колоната тип). При прилагане на филтриране се запазват само редовете, за които условието е истина;
* премахване на дублиращи се редове.

За колоните са възможни следните операции:

* премахване на дадена колона (по име или номер);
* промяна на реда на колоните - с подаване на нова пермутация на номерата на колоните. Например за таблица с три колони, пермутацията 2 3 1 означава, че първата колона ще стане последна.

Всяка от гореизброените операции трябва да може да се отменя (команда **undo**). В този проект се изисква възможност за отмяна само на последната приложена операция.

В края на таблицата трябва да има възможност за добавяне на нов ред, който да представлява едно от:

* копие на ред с подаден номер;
* ред, съставен от минималните стойности за всяка колона;
* ред, съставен от максималните стойности за всяка колона;
* ред, съставен от най-често срещаните стойности за всяка колона. Ако има повече от една с еднаква честота, да се използва най-малката от тях.

# Тема №4 Социална мрежа

В рамките на този проект трябва да се реализира конзолно приложение, което наподобява социална мрежа. В социалната мрежа потребителите могат да задават въпроси от различен характер, които са групирани по теми. Всички данни за мрежата трябва да се съхраняват във файлове (команди **save**, **save as**) и да се зареждат при стартиране на програмата (команда **load (т.к. open е заето от друга команда)**). Ако няма валидни файлове, то се стартира с празни данни. Имате свободата сами да определите формата на файловете, но трябва да опишете този формат, както и да подсигурите коректност на входа.

Операциите, които трябва да поддържате са следните.

### Създаване на потребителски акаунт.

В началото всеки потребител трябва да създаде свой акаунт. За тази цел се използва командата **signup.**

Всеки потребител се характеризира с:

* собствено име;
* фамилно име;
* потребителско име (уникално за системата);
* парола;
* уникален идентификатор (неотрицателно число), който се генерира при създаването на потребителския акаунт;
* точки.

Командата изисква последователното въвеждане на потребителско име и парола, последвани от собствено и фамилно име. Броят точки на всеки новосъздаден потребител е 0.

Потребителите могат да са в една от двете роли - обикновен потребител или модератор. Първият създаден в системата потребител е модератор, останалите се създават като обикновени потребители.

### Вход в съществуващ потребителски акаунт

Ако даден потребител вече има създаден акаунт, може да влезе свободно в системата, като използва командата **login**. Командата изисква въвеждането на потребителско име и парола. Ако е намерено съответствие, потребителят успешно влиза в своя акаунт.

### Промяна на потребителски акаунт

Всеки потребител трябва да има възможност за промяна на данните в собствения си акаунт (команда **edit**). Променят се имената или паролата. Ако потребителят е модератор, той може да променя и чужд акаунт (**edit id**), като освен имената и паролата има право да променя и ролята (да дава или да отнема модераторски права).

Основната функционалност на системата е задаване и отговаряне на въпроси. Въпросите са групирани по теми. Всяка тема се характеризира със:

* заглавие;
* идентификатор на създателя;
* кратко описание на темата;
* уникален идентификатор на темата (неотрицателно число), който се генерира при създаването на темата.

Операциите, свързани с теми са следните.

### Създаване на нова тема

Потребител, успешно влязъл в своя потребителски акаунт, може да създаде нова тема с командата **create**. Командата изисква въвеждане на заглавие и кратко описание на темата.

### Търсене на тема

Търсенето на съществуваща тема се осъществява с командата **search**. Командата изисква въвеждане на част от заглавие на тема.  
Резултатите се извеждат на нов ред във формата<title> {id: id}.

Пример:

**> search oop**

>> oop in C++ {id: 42}

>> helping with oop {id: 47}

>> design patterns making oop simpler {id: 59}

### Зареждане на тема

Зареждането на съществуваща тема се осъществява с командата **open**. Тя изисква въвеждането на пълното заглавие на темата или на нейния идентификационен номер.

Ако командата е успешна, на стандартния изход се извежда кратка информация за темата, а ако не е - подходящо съобщение.

### Извеждане на всички въпроси в отворената тема

За да се изведат всички въпроси (дискусии) в една отворена тема, се използва командата **list**. Командата не изисква допълнителни параметри.

Пример:

**> open 42**

Welcome to “oop in C++”.

**> list**

What is an abstract class? {id: 77}

Virtual methods? What do we need them for? {id: 90}

### Добавяне на въпрос в отворената тема

Всеки въпрос (дискусия) към дадена тема се характеризира със:

* заглавие;
* съдържание на въпроса;
* идентификатор на потребителя, който го е задал
* уникален идентификатор в рамките на темата (неотрицателно число).

Добавяне на въпрос към отворена тема се осъществява с командата **post**. Командата изисква въвеждане на заглавие и съдържание на въпроса.

### Разглеждане на въпрос (дискусия) в отворената тема

За да се разгледа съдържанието на даден въпрос (дискусията към него), се използва командата **post\_open**. Командата изисква въвеждането на неговия идентификационен номер.Ако командата е успешна, на стандартния изход се извежда пълното заглавие на въпроса, съдържание на въпроса и броя негови отговори. В противен случай да се изведе подходящо съобщение.

След като даден въпрос е отворен, за него могат да се извършат следните действия:

* да се добави коментар;
* да се отговори на съществуващ коментар;
* да се реагира с upvote или downvote на някой коментар (всеки потребител може да реагира само веднъж на даден коментар);
* премахване на коментар (може да се извърши само от автора на коментара или модератор)
* да се изведат всички коментари и техни отговори на въпроса.

Всеки коментар се характеризира с:

* автор;
* текст на коментара;
* оценка (цяло число със знак - сума от всички положителни и отрицателни реакции);
* уникален идентификатор на коментара в рамките на въпроса.

Не е проблем да съществуват два коментара с еднакви идентификатори, стига да се отнасят за два различни въпроса. Оценката на коментар се добавя към оценката на автора му.

### Затваряне на текущ въпрос, текуща тема

Четенето и писането на коментари по текущия отворен въпрос приключва с изпълнение на **post\_quit.** Командата не изисква допълнителни аргументи. Текущата отворена тема се затваря с командата **quit**.

### Премахване на въпрос или тема

Само модераторите могат да премахват въпрос или цяла тема. При премахване на въпрос се премахват всички коментари в него, съответно при премахване на тема се премахват всички въпроси в нея.

При премахване на коментар, въпрос или тема оценките на потребителите, чиито коментари се премахват трябва да бъдат съответно модифицирани.

### Изход от потребителски акаунт

Командата **logout** се използва за изход от потребителския акаунт. Командата не изисква допълнителни аргументи

### Затваряне на приложението

При затваряне на приложението (командата **exit**) всички направени промени трябва да бъдат записани на диска.

# 

# Тема №5 Приложение за работа с електронни таблици

В рамките на този проект трябва да се реализира конзолно приложение, което работи с електронни таблици.

### Представяне на данните

Данните на една таблица се записват 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 стойности 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 '-'.

Например: 123.456; -123.456; +123.456.

**Символен низ (стринг)** – поредица от произволни символи, оградени в кавички. Подобно на низовете в C++, ако в даден низ трябва да бъде включен символа за кавичка, то той трябва да бъде представен като \". За да бъде включена наклонена черта, тя трябва да бъде представена 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 (/), сравнение (==, !=, <, >, <= или >=). 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 изчислена за дадената клетка. Повече информация за формулите е дадена по-долу.

### Нужна функционалност

Програмата трябва да може да зарежда (open), записва (save, save as) и затваря (close) една таблица.

След като приложението отвори даден файл, то трябва да може да:

* извежда съдържанието на таблицата на екрана (print);
* редактира съдържанието на дадена клетка (edit),

Ако при зареждането на данните, приложението ви открие грешка, то трябва да изведе подходящо съобщение за грешка. 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 запетая.
* 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.

### Извеждане на таблицата на екрана

**> print**

Командата **print** се използва за извеждане на заредената таблица на стандартния изход. 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 <row> <column> <new\_value>**

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 трябва да бъде променена, а също и каква стойност да се запише в нея. Потребителят може да въведе произволен тип данни, който се поддържа от програма (например цяло число, дробно число, низ, формула и т.н.).

Ако потребителят въведе неправилни данни, приложението не трябва да променя нищо в таблицата, а само да изведе на стандартния изход подходящо съобщение за грешка. Забележка! Командата може да промени типа на указаната клетка.

### 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.

Като аргументи в дадена формула могат да участват единствено:

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

При операциите важат следните правила:

1. Ако в дадена формула участват само числа, то операцията се извършва по традиционните правила на аритметиката. Логическите операции се оценяват като 0 или 1. Специален случай е делене на две цели числа. В този 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 преобразува 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 |

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

Нека е дадена примерна таблица, в която 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 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 | „Hello world!“ се преобразува до нула |
| = R1C1 == R1C2 | 10 == 0 | 0 | „Hello world!“ се преобразува до нула |
| = R1C1 <= R2C1 | 10 <= 123 | 1 | Низът „123“ се преобразува до 123. |
| = R1C1 \* R2C1 | 10 \* 123 | 1230 | Низът „123“ се преобразува 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. Счита се, че тя е празна. |
| = 10 / 0 | 10 / 0 | ERROR |  |
| = 10 / R1C2 | 10 / 0 | ERROR |  |
| = R1C1 / R1C2 | 10 / 0 | ERROR |  |

# 

# Тема №6 JSON парсер

В рамките на този проект трябва да се реализира конзолно приложение, което работи с файлове в JSON формат.

[*JSON*](https://www.json.org/json-bg.html) *е популярен текстов формат за описание на данни, взаимстван от синтактичните правила на езика за програмиране JavaScript за деклариране на обектни литерали.*

***Забележка:*** За решението на задачата не е позволено използването на готови библиотеки за работа с JSON. Целта на задачата е да се построи подходящо представяне на данните в паметта и да се реализират операции с него.

Допускат се улеснения на задачата при прочитане на входа, например да не се поддържат нови редове, кавички, или специални символи в символните константи и др. Опишете какви улеснения сте допуснали в документацията към проекта.

При подаване на валиден JSON файл програмата трябва да го прочете и съхрани в подходящо подбрана от вас структура в паметта. С получения обект да могат да се извършват следните операции, в допълнение на общите операции (open, close, save, save as и exit):

|  |  |
| --- | --- |
| validate | Да се направи проверка дали отвореният 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> | Програма трябва да провери дали в обекта се съдържат данни, записани под този ключ. Ключът може да е на произволно ниво на вложение в JSON документа. Ако такива данни са открити, да се изведе масив от всички такива данни (т.е. всички срещания на търсения ключ), като всяка от е представена като JSON обект.  Бонус: Търсенето да става по регулярен израз (например “person-\*” търси всички ключове, започващи с израза “person-”). |
| contains <value> | Програмата трябва да провери дали в обекта има стойност, която съдържа <value>. Стойността може да е на произволно ниво на вложение в документа. Съблюдавайте типовете на данните. Например стойността 19 се съдържа в данните от символен тип “Born in 1998”, но не и в числото 1998. |
| set <path> <string> | При подаване на пълен път към даден елемент и символен низ, програмата трябва да замени стойността на посочения елемент с прочетената 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. |
| create <path> <string> | При подаване на пълен път към даден елемент и символен низ, програмата трябва да добави такъв елемент и да му присвои прочетената според синтаксиса на JSON стойност на низа. Ако такъв елемент съществува, трябва да се обяви това като проблем. В противен случай, да се добави елементът, евентуално с рекурсивно създаване на целия път. Ако символният низ не е коректен, 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 той съществува. Ако <path> не е подаден, да се записва целият обект, който в момента е зареден в паметта. |

Например, нека имаме следния JSON файл, описващ организация.

organisation.json

|  |
| --- |
| {  "name":"ACME Software Co.",  "type":"Software Development Company",  "offices": [  {  "name":"Headquarters",  "address":"Sofia"  },  {  "name":"Front Office",  "address":"New York City"  }  ],  "members":[  {  "id" : 0,  "name" : "John Smith",  "birthdate" : "1980-01-01"  },  {  "id" : 1,  "name" : "Jane Smith",  "birthdate" : "1981-02-02"  },  {  "id" : 2,  "name" : "John Doe",  "birthdate" : "1982-03-03"  }  ],  "management":{  "directorId":0,  "presidentId":1  }  } |

Следват няколко *примерни* команди и резултатите от тях:

|  |
| --- |
| search "name"  [  "ACME Software Co.",  "Headquarters",  "Front Office",  "John Smith",  "Jane Smith",  "John Doe"  ]  set "management/directorId" "2"  search "management"  "management":{  "directorId":2,  "presidentId":1  } |

# 

# Тема №7 Бази от данни

В рамките на този проект трябва да се реализира приложение, което поддържа операции с прости бази от данни.

Базите от данни се състоят от серии от таблици, като всяка таблица е записана в собствен файл. Базата от данни е записана в главен файл (каталог), който съдържа списък от таблиците в базата, като за всяка таблица е зададено име и файл, в който таблицата е записана.

### Поддържани типове данни

Всяка “колона” на таблица в базата от данни има тип, като в една таблица може да има едновременно колони от различни типове. Вашето приложение 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 (стринг)** – поредица от произволни символи, оградени в кавички. Подобно на низовете в C++, ако в даден низ трябва да бъде включен символа за кавичка, то той трябва да бъде представен като \". За да бъде включена наклонена черта, тя трябва да бъде представена 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 дадена база, чрез нейния основен 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 добавяне на таблица с име, което съвпада с името на някоя вече съществуваща таблица, системата да дава грешка. Добавената таблица се записва в каталога на базата от данни. |
| --- | --- |
| showtables | Показва списък с имената на всички 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 съдържанието на таблица в текстов файл, форматирано по подходящ начин.. |
| 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. |
| select\_onto <result\_table> [<comumn\_1>, <column\_2>, ...] <column-n> <value> <table name> | Подобно на select, но резултатът се записва в нова таблица с името result\_table, и съдържа само колоните с номера изброени в списъка, ограден от []. В този списък номерата могат да се повтарят и да са в произволен ред. Този ред трябва да се запази в резултатната таблица. |
| 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. |

# 

# Тема №8 Обработка на растерни изображения

В рамките на този проект трябва да се разработи приложение, което представлява конзолен редактор на растерни изображения. Вашият 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 трябва да може да работи с PPM, PGM, PBM файлове (за повече информация, вижте<http://en.wikipedia.org/wiki/Netpbm_format>).

Желателно е да реализирате поддръжка на BMP изображения (поне в основния профил на формата). За справка: <https://en.wikipedia.org/wiki/BMP_file_format>.

Вие трябва да реализирате кода за четене / запис на изображенията.

### 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 с 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 случай на множество потребителски сесии.

Пример:

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

Session with ID: 1 started

Командата стартира сесия, в която ще се обработват изображенията image01.ppm и image02.pgm. Когато се прилагат трансформации в сесия, те важат за всички заредени изображения за **текущата** сесия.

### Конвертиране до черно-бяло изображение

Команда:

**> grayscale**

Командата трябва да бъде приложена 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 или BMP), но да бъде чернобял, т.е. всички пиксели в него да са черни, бели или нюанси на сивото.

### Конвертиране до монохромно изображение

Команда:

**> monochrome**

Поведението на тази операция е като това на предишната, но тук файлът се конвертира до монохромен, т.е. 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

Команда:

**> negative**

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

### Завъртане на изображение

Команда:

**> rotate left**

или

**> rotate right**

или

**> flip top**

или

**> flip left**

Тези команди реализират завъртане и получаване на огледален образ на изображение. Обърнете внимание, че четири последователни ротации в една посока са идентитет, следователно няма нужда да бъдат прилагани. Същото се отнася и за две отражения. Съществуват и други серии трансформации, резултатът от прилагането на които е идентитет и няма смисъл те да се прилагат.

### Отказ от последна трансформация

Команда:

**> undo**

Командата **undo** 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.

Команда:

**> redo**

Командата **redo** възстановява последно отменената с **undo** трансформация в текущата сесия. Ако няма отменена трансформация или всички отменени вече са възстановени, то тя не трябва да има никакъв ефект.

### Добавяне на изображение

Команда:

**> add** image02.ppm

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

### Изрязване на изображение (crop)

Команда:

**> crop 20 10 1000 150**

Командата получава координатите на горен-ляв и долен-десен връх на правоъгълник и изрязва всички части от изображенията в текущата сесия, които не влизат в него. Ако правоъгълникът излиза извън рамките на изображение, той автоматично да се смалява до неговите рамки. Например, ако изображението е с размери 100x100 и искаме да изрежем правоъгълника с горен-ляв ъгъл в точката (20,10) и долен-десен ъгъл в (1000,150), ще остане само частта от изображението, заключена между (20,10) и (99,99).

### Запис на промените

Команда:

**> save**

За запис на направените трансформации е необходимо да реализирате команда

**save**. Тя записва новосъздадените файлове в същата директория като оригиналните изображения в сесията. Към името на файловете се добавя времето на тяхното записване.

Пример:

**> load**image01.ppm

Session with ID: 1 started

**> grayscale**

**> save**

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

Команда:

**> saveas**

За запис на направените трансформации може да се използва команда

**saveas**. Тя записва новосъздадените файлове в същата директория като оригиналните изображения в сесията. Тук имената са подадени като параметри на командата и **трябва да са не повече** от броя на заредените в сесията изображения. Първото име се прилага за първия зареден файл, второто за втория и т.н. Ако подадените имена са по-малко от файловете в сесията, за всички, за които не е подадено име се използва служебно име (виж **save**).

Пример:

**> load**image01.ppm somePic.bmp

Session with ID: 1 started

**> grayscale**

**> saveas grayfile.ppm changedPic.bmp**

Files grayfile.ppm and changePic.bmp successfully saved.

### Информация за текуща сесия

Команда:

**> list session**

Командата дава възможност на потребителя да получи подробна информация за текущата потребителска сесия: нейния идентификационен номер и участващите в нея изображения - техните имена и формати.

Пример**:**

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

Session with ID: 1 started

**> rotate left**

**> grayscale**

**> session info**

Files in session with ID 1:  
image1 - portable pixmap format

image2 - portable graymap format

image3 - 32bit Bitmap

### Излизане от програмата

Команда:

**> exit**

Излизането от графичния редактор трябва да се осъществява чрез командата **exit**. Ако има незаписани трансформации трябва да попитате потребителят дали иска да ги запази преди излизане.

Пример:

**> load** image.ppm

**> negative**

**> exit**

Session contains unsaved work. Save it now (y/n)?

**> n**

В този случай не трябва да се създава нов файл и да се прилага трансформацията **negative**, защото преди **exit** не сме извикали **save** и сме отказали записване при излизане.

Имплементирайте функционалност, която да поддържа създаването на колаж от две изображения, в един и същ формат, които са налични в текущата сесия. Командата е **make collage**.Модификаторите **vertical** и **horizontal** са допълнителни опции към нея**,** въз основа на които изображенията се долепят едно над друго или едно до друго, респективно. Ако по съответното измерение размерите са различни, по-малкото изображение да се разшири, като се допълнят нови редове (колони) с черен цвят. Изображението да е центрирано спрямо добавените редове (колони).

Пример:

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

**> make collage horizontal** image1.ppm image2.ppm

**> make collage vertical** image1.ppm image3.pgm

Cannot make a collage from different types! (.ppm and .pgm)

**> save**

Създава се файлът **image1\_image2\_<current\_time>.ppm**, резултат от хоризонталното долепяне на двете изображения image1.ppm и image2.ppm, при положение, че тяхната размерност съвпада и изображенията съществуват, заредени в текущата сесия.