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

КУРСОВ ПРОЕКТ ПО ОБЕКТНО-ОРИЕНТИРАНО ПРОГРАМИРАНЕ

летен семестър 2019/2020г.

Проект 17 – Растерна графика

Изготвил: Атанас Миленов Иванов

Специалност: Информационни системи I курс

Факултетен номер: 71937

Група: 3

Ръководител на курс: Калин Георгиев

Юни 2020

гр.София

**Глава 1. Увод**

1.1. Описание и идея на проекта……………………………………………..…………..

1.2. Цел и задачи на разработката ……………………………………………………….

**Глава 2. Преглед на предметната област**

2.1. Основни дефиниции, концепции и алгоритми…………………………………….

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**

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**

4.1. Реализация на класове ----------------------------------------------------------------------

4.2. Управление на паметта и алгоритми. Оптимизации.------------------------------------

4.3. Планиране, описание и създаване на тестови сценарии --------------------------------

**Глава 5. Заключение**

5.1. Обобщение на изпълнението на началните цели-----------------------------------------

5.2. Насоки за бъдещо развитие и усъвършенстване-------------------------------------------

**Глава 6. Използвана литература**

За създаването на диаграмите:

<https://app.diagrams.net/>

За информацията свързана с форматите:

<https://en.wikipedia.org/wiki/Netpbm#File_formats>

**Важно:** НЕ са използвани std::vector и std::string, а са употребени ръчно написани.

1. Увод

**1.1 Описание и идея на проекта**

Проектът „Растерна графика“ е свързан с обработването и редактиране на снимки в различни формати чрез конзола. Приложението може да работи с файлове, да стартира сесия, да прилага различни трансформации върху изображенията и записване на резултата. Валидните формати са **PPM**, **PGM** и **PBM**.

**1.2 Цел и задачи на разработката**

Основната цел на проекта е да дава възможност на потребителя да работи лесно с различни видове снимки, като се използват лесни за разбиране и управление команди. Това би било особено полезно за човек, който обича да се занимава с редакция на изображения или пък при нужда от извършване на основни трансформации.

Основните функционалности са:

* Трансформация на изображение до нюанси на сивото (grayscale)
* Трансформация на изображение до монохромно (monochrome)
* Негатив върху изображение (negative)
* Ротиране в посока (rotate)
* Връщане на вече направена операция (undo)
* Добавяне на изображение в текущата сесия (add)
* Информация за сесията, участващи изображия, и набора от трансформации, които предстоят да бъдат направени (session\_info)
* Колаж от две изображения в трето, която се добавя в сесията (collage)

Основните компоненти са Image (с наследници PGM, PPM, PBM), Session, ConsleEngine.

**Image –** абстрактен клас, който съдъръжа:

Магическо число (magicNumber)

Колони и редове (cols, rows)

Матрица от пиксели (pixelMatrix)

\* в добавка, форматите PGM и PPM имат и максимална стойност (maxValue)

**Session** съдържа:

Идентификационен номер (ID)

Трансформации (Vector<String> transformations)

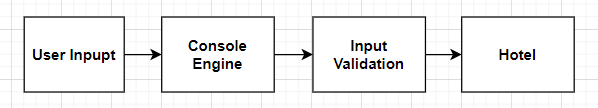
Директории (Vector<String> directories)

Снимки (Vector<Image\*> images)

1. Преглед на предметната област

**2.1 Основни дефиниции, концепции и алгоритми**

Основният компонент който задвижва програмата е ConsoleEngine, който позволява нашата система да работи като конзола, която изпълнява операции подадени на един ред. Потребителят има право да въвежда колкото си иска команди, но конзолата ще работи само при правилно въведени такива.



Фиг.1 Как програмата обработва входа на потребителя

При зареждане на приложението, се отваря конзола, която очаква вход от потребителя или по-скоро валидна операция (една от изброените в глава 1.2). Нека вземем за пример операцията **availability**. Потребителят въвежда “rotate right” (на един ред) и както в една обикновена конзола, тя разбира че иска да се изпълни операцията rotate с единствен параметър right. На фиг. 1 се вижда нагледно как става процесът. Първоначално входа на потребителя представлява един символен низ, който чрез помощта на функциите в ConsoleЕngine бива обработен по подходящ начин. Става дума за следните функции:

* **countLetters**
* **indexOf**
* **substring**
* **split**
* **toInt**

Използвайки ги, въведеният символен низ се разделя съответно на отделни думи, които първо биват валидирани и след това извикват правилните функции от хотела. Подробното им действие е описано в следващата глава.

Също така много важен за правилното действие на апликацията е принципът на **наследяването**, чрез който при зареждане на снимка става ясно от какъв тип точно е тя (ppm, pgm или pbm). Описано е подробно в 2.2.

**2.2 Дефиниране на проблеми и сложност на задачата**

Първият проблем, който възниква, е именно разделянето на символния низ на отделни думи, както беше споменато горе. Тук идва на помощ функцията **split**, която е известна и в други програмни езици (разделя символен низ на отделни думи). Следващият въпрос е как програмата разбира коя операция да изпълни? Чрез един switch блок, и с **checkOperation** става ясно коя точно операция трябва да бъде използвана. За правилната работа на ConsoleEngine, снимките, с които се работи са инициализирани в **ConsoleEngine - run()**.

Важно е да се спомене проблема с невалидния вход от потребителя. Програмата трябва да разбира кога някой се опитва да въвежда грешни неща и съответно или да го предупреждава или да спира действието и, но да не предизвиква bug-ове или crash-ове.

Нека първо разгледаме няколко невалидни входа:

- Подаване на текст, там където се иска число

- Подаване на повече или по-малък брой параметри отколкото е нужно

- Подаване на грешна команда

- Оставяне на повече място от нужното между 2 от аргументите или неоставяне на никакво място между тях

- Въвеждането на невалидна дата (може би най-сложната валидация в приложението)

Ако се стигне до някой от тези случаи, потребителят получава съобщение за грешка, което не прекратява програмата, а му позволява да въведе правилна команда.

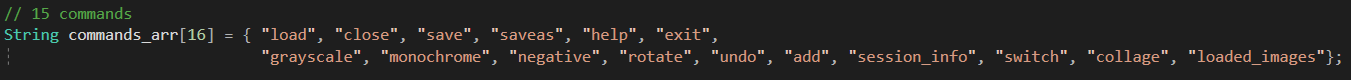
От критично значение беше справянето с йерархичната структура на класовете и справянето с наследяването. От много хубава полза беше употребата на указатели за самите снимки при създаването им както и после в сесията. По този начин принципът на **полиморфизма** даде отличен резултат. Също и как да работи матрицата от пиксели беше един от по-сложните проблеми, затова беше използван **Vector<Vector<size\_t>>,** който играе роля на двоичен масив.

**2.3 Подходи, методи за решаване на проблемите**

https://i.gyazo.com/0299040e4e1dfff850a4240d9571be10.pngНека първо видим как става разделянето на отделни думи при въвеждането на символния низ.

Фиг.2 Команда за разделяне на символен низ

Командата приема символен низ и ни връща вектор от символни низове. Това е доста удобно, понеже ще имаме достъп до всяка една данна, която е необходима за програмата да разбере коя операция да извърши. А как всъщност се решава това?

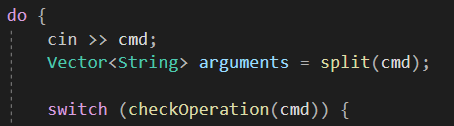


Фиг.3 Всички валидни операции

https://i.gyazo.com/d7b405644e8708c064fdc4b6b5912cc1.png

Фиг.4 Фунцкия за определяне на правилната команда

Използвайки функцията **checkOperation** ние разделяме със split подадения символен низ на вектор от думи и проверяваме дали първият елемент на този вектор отговаря на някоя от командите от Фиг.3. Ако съвпада ние връщаме число което е с едно по-голямо от позицията на тази команда в масива от Фиг.3. Така влизаме в някои от слуачите на switch блока в командата **ConsoleEngine::run()**.



Фиг.5 Задействането на run()

Нека сега видим някои от валидациите, които предотвратяват грешен вход от потребителя. Като за начало, ако входът на потребителя НЕ започва с някоя от операциите излиза грешка. Във всяка от отделните валидации има проверка за разстоянието между думите така че то също да бъде валидно.

Следва проверката за правилно въведено изображение. С помощта на функциите **checkImageFormat** и **matchFormat** проверяваме дали файлът е с правилно разширение и ако то не е, излиза грешка на потребителя. Обратно пък, ако е правилен формата, снимката получава formatID, което се използва в matchFormat. Идеята е да съвпадне с някои от форматите за правилното инициализиране на снимката. https://i.gyazo.com/8667f0208a899c055aafb21a11ed71b4.png

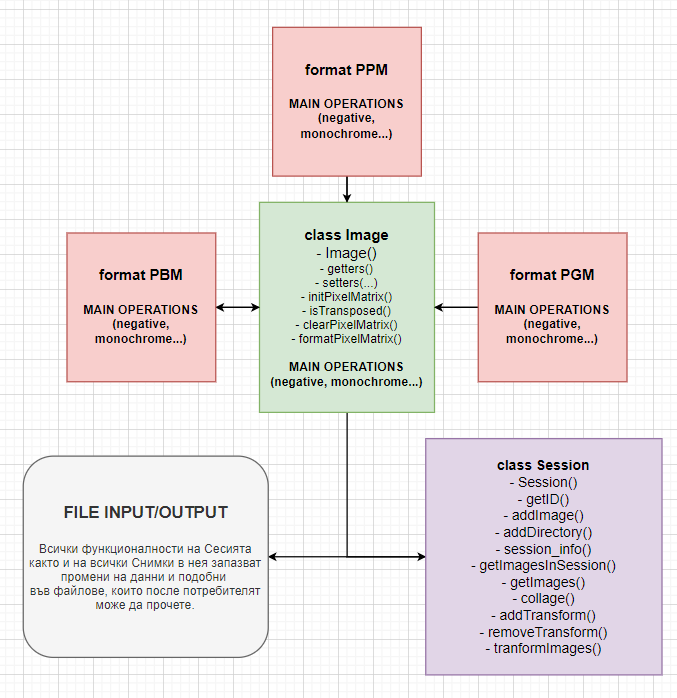
https://i.gyazo.com/65c3667768bcd14efdace30f1020bc4e.png

Фиг.6 Използването на двете функции

Така програмата ще продължи само ако изображението съответства на някой от форматите.

1. Проектиране

**3.1 Обща архитектура – ООП дизайн**

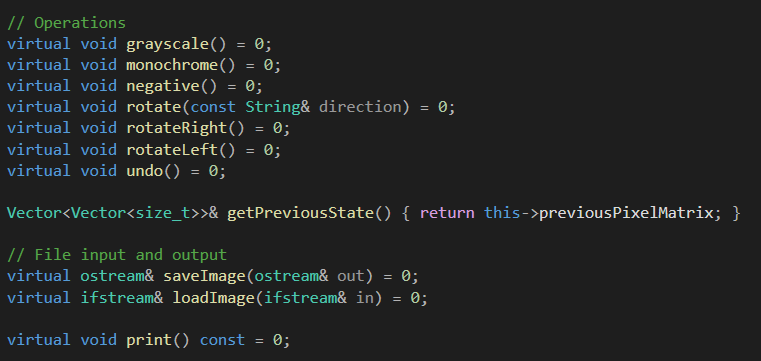
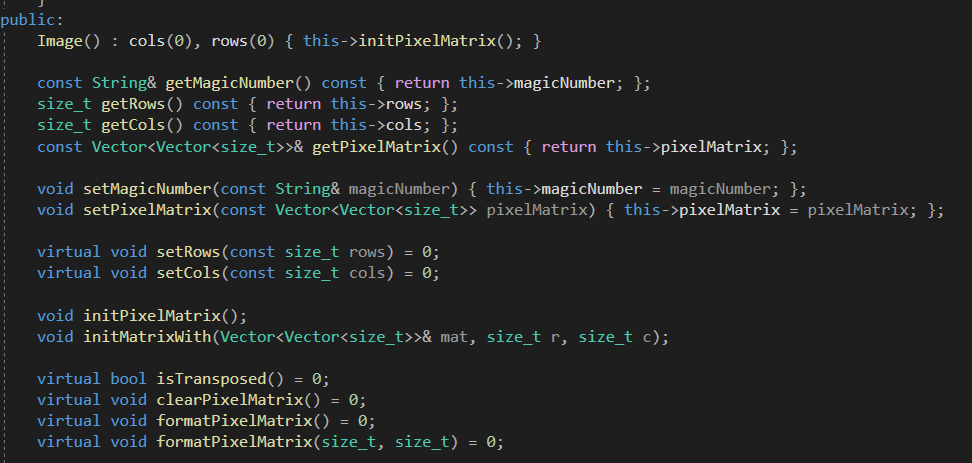


Фиг.7 Архитектура на проекта

На фигурата са показни основните класове, методите в тях и как взаимодействат помежду си. Тук не беше нужно да се реализират толкова класове, колкото време беше необходимо за ефективни дефиниции на самите методи.

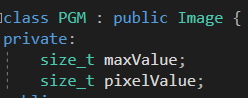
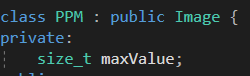
**3.2 Структури и поведение**

Извадките от кода са от главния клас Image и Session, които вършат основната работа тук.



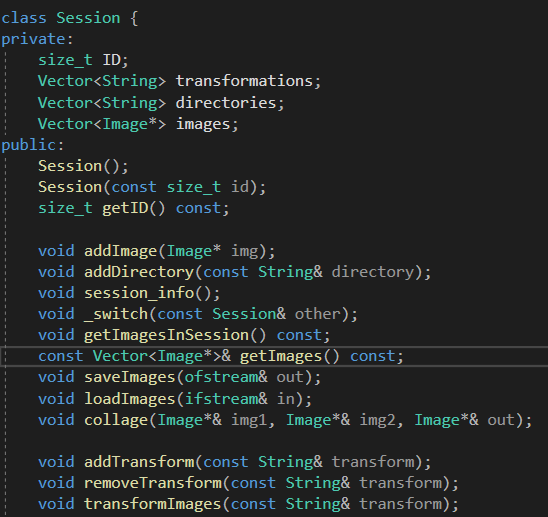
Фиг.8 Декларации на методите в Image

Както споменах, има нужда от доста методи тук. Разбира се, наследниците на класа PGM, PBM и PPM имат всички тези (реално тяхните дефиниции са там) и така класът Image е абстрактен.

https://i.gyazo.com/98f8cbe4ae3ea54d70f709f55738226b.png

Фиг. 9 Наследниците на Image

Когато потребителят зареди изображение с **load(open)** се създава потребителска сесия. Това изображение се добавя към нея и тя става основният двигател на програмата. Тези класове наследници от фиг.9 се използват там.



Фиг.10 Класът на сесия

Доста значителни са двата вектора **transformations** и **directories**. Векторът от трансформации съдържа като символни низове всички направени **трансформации върху изображенията в текущата сесия**. По този начин може да се даде правилна информация на потребителя при използването на session\_info, както и при други функционалности. Векторът от директории съдържа отново като символни низове **директориите на добавените снимки в сесията**. Те от своя страна са много полезни при записа на и при четенето на всички изображения по правилен начин. Така когато запазваме например всички изображения в сесията се отваря поток към файл с име съответната директория.

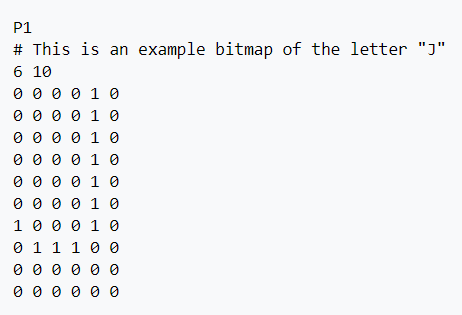
За правилната работа с всички файлове (изображения), всеки от форматите си има **save** и **load** методи, които служат за четенето от файл и записването в него. Сесията също има saveImages и loadImages, които обхождат вектора от снимки и викат съответно правилните методи.

1. Реализация, тестване

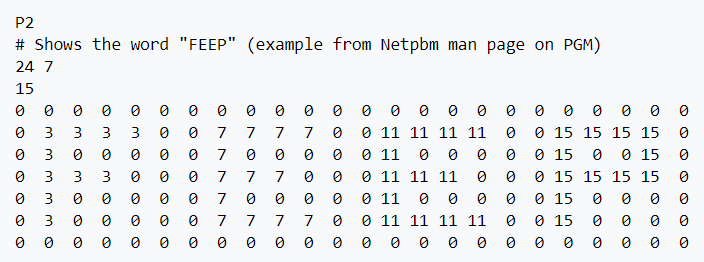
**4.1 Реализация на класове**

Нека видим реализацията на функционалностите на изображенията. Тъй като всяка функция имаше различна реализация за съответния формат ще бъде показана само на един от трите формата. Принципът в другите формати е подобен. Ще започнем с това как всъщност изглеждат снимките от този формат.

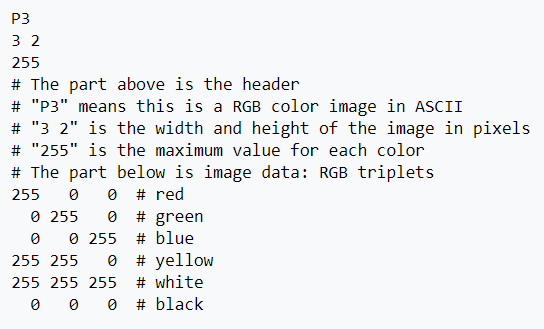
PBM е формат, който съдържа само нули и единици като пиксели и няма максимална стойност на пиксела. PPM и PGM имат максимални стойности, които определят най-яркия нюанс в изображението. PGM съдържа нюанси на сивото като 0 отговаря на черния цват, а максималната стойност отговаря на белия цвят.



**PBM**

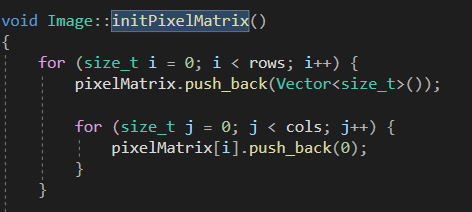


**PGM**



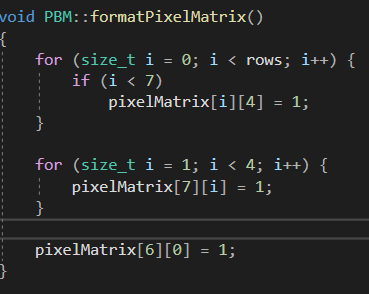
**PPM**

Така изглеждат различните формати като матрица от пиксели. Нека видим тяхната инициализация като код.

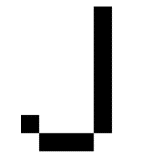


Фиг.11 Инициализация на изображенията

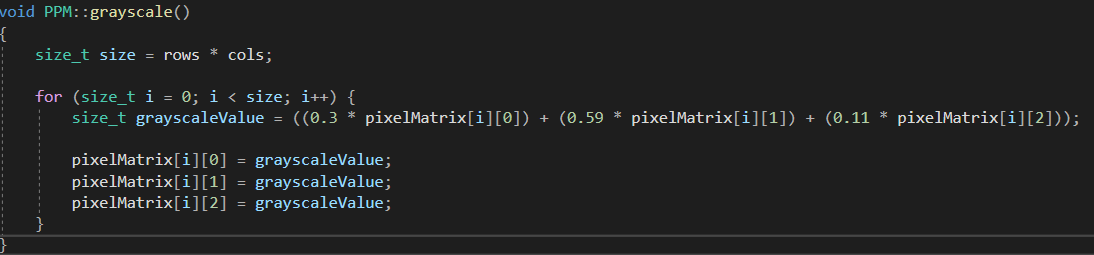
Тук се създават доста проблеми с паметта, ако не е направено по подобен начин. Идеята е, че на всеки ред матрицата трябва да добавя един празен вектор от size\_t, така че да има къде да добави нулите от втория цикъл. В противен случай на местата, където трябва да се добавят (например pixelMatrix[0][1]) не просто няма елемент, ами и не е инициализирано нищо. Затова се налага извикването на **push\_back(Vector<size\_t>())** на всеки ред. За да придадем вид на изображението, беше създадена функция **formatPixelMatrix**, която вършеше точно това – форматиране на снимката.



Фиг.12 Форматиране на формата PBM

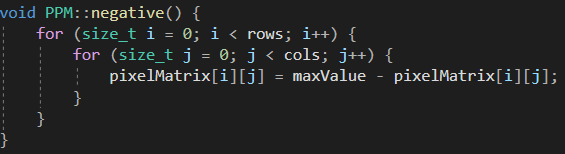
Така получаваме образ на буквата “J” в PBM формат. Важно е да се отбележи, че това е приближено 20 пъти.

Ще обърнем внимание на операциите **grayscale** и **negative**.



Фиг.13 Grayscale на формата PPM

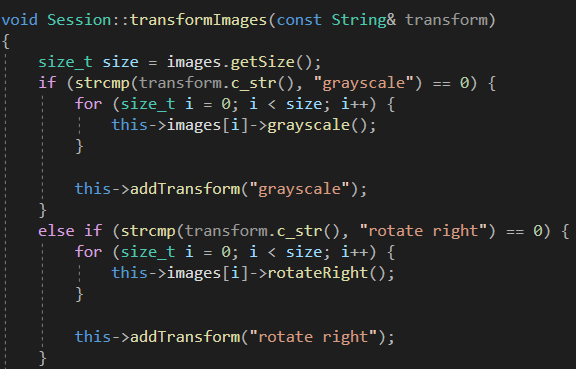
Този формат е подходящ за тази функция, а и е най-приложим тук, тъй като тук участват RGB triplets – тройки от червено, зелено и синьо, които образуват цвят от тяхната комбинация. При прилагането на формулата в grayscaleValue, изображението се трансформира само до нюанси на сивото. Съответно се променят и пикселите. Другите два формата нямат ефект от тази функция, понеже те по начало съдържат само нюанси на сивото.



Фиг.14 Негатив на формата PPM

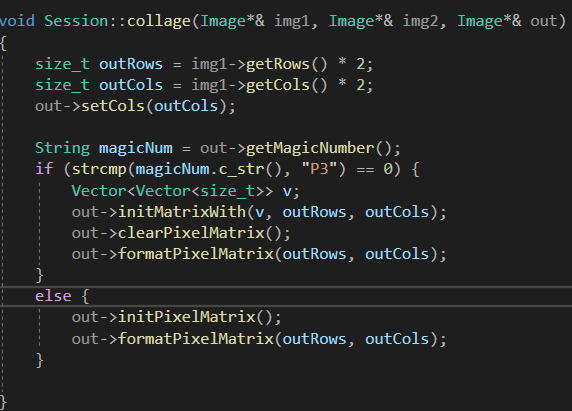
Операцията негатив е по-обща и за трите формата. Тя прави негатив на заредената снимка, като от максималната стойност вадим текущата стойност на пиксела в матрицата. Ако тази функция бъде извикана два пъти, изображението се връща в първоначалния си вид.

Като за финал ще разгледаме и сесията (на фиг.10 се виждат дефинициите на методите в нея). Да разгледаме как работят функциите **transformImages**, **collage** както и как се забазват изображенията във файловете.



Фиг.15 Прилагане на трансформация в сесията

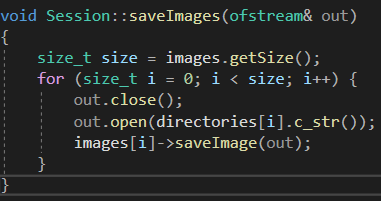
Операцията получава единствен параметър – символен низ, който дава името на трансформацията. Задейства се един дълъг if-else statement, който има за цел да пасне трансформацията с валидна такава. Ако тя съвпадне, обхождаме векторът от снимки в сесията и на всяка една прилагаме съответната трансформация. Обратно, ако символният низ не съответства на нито една правилна функция, излиза грешка.



Фиг.16 Колаж на две снимки

При извикването на колаж се приемат като параметри двете снимки, които искаме да обединим и тази, в която искаме да ги запазим. Тя се добавя автоматично в текущата сесия или може да бъде заредена предварително. Удвояваме колоните, поради факта че искаме да получим удвоено изображение, следователно ни трябва по-голяма матрица от пиксели. If-а е нужен поради различната инициализация на формата с магическо число „P3”, а именно PPM. Главната идея е да се инициализира снимката **out**, както и да се форматира по правилния начин. Ако форматът е PPM, се еналага първо да изчистим матрицата (да направим всички стойности 0) преди да я форматираме.

Не на последно място е правилното записване на изображенията в сесията. За постигане на този резултат е използвана **saveImages**.



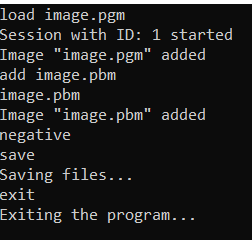
Фиг.17 Запазване на снимките във файлове

Тъй като всяка снимка се различава от друга по нейното име (или по „посоката“ на потока, който трябва да бъде отворен към файла) ни е необходим векторът от трансформации, както беше споменато по-рано в документацията. В началото на всяко завъртане на цикъла първоначално подадения като параметър поток **out** бива затворен и се отваря такъв към правилната директория, където искаме да запазим снимката. След това се извиква saveImage към същото място на всяка снимка от тези в сесията. Подобно е действието и на **loadImages**.

**4.2 Управление на паметта и алгоритми**

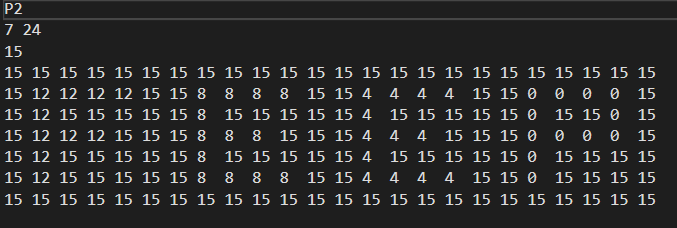
Динамичната памет се контролира най-вече от Vector и от String, които са ръчно написани и се справят доста добре именно с това. Имаше проблеми, които се появяваха по време на тестване на програмата, но бяха изчистени във методите най-вече на вектора. Тъй като снимките, които биват зареждани от потребителя, се ползват във **Vector от снимки** в сесията, имаме привилегията класът вектор да се грижи за управлението на паметта в heap-a.

**4.3 Планиране, описание и създаване на тестове**

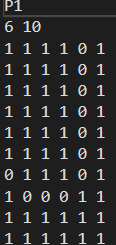


При зареждането на снимка се зарежда потребителската сесия и снимката се добавя в нея. Можем чрез командата add да добавявме още снимки в сесията. Естествено ако не добавим аргумент ще излезе съобщение за грешка на всяка операция.

Прилагаме негатив и запазваме файлове. Трансформациите се прилагат на всички снимки в сесията.



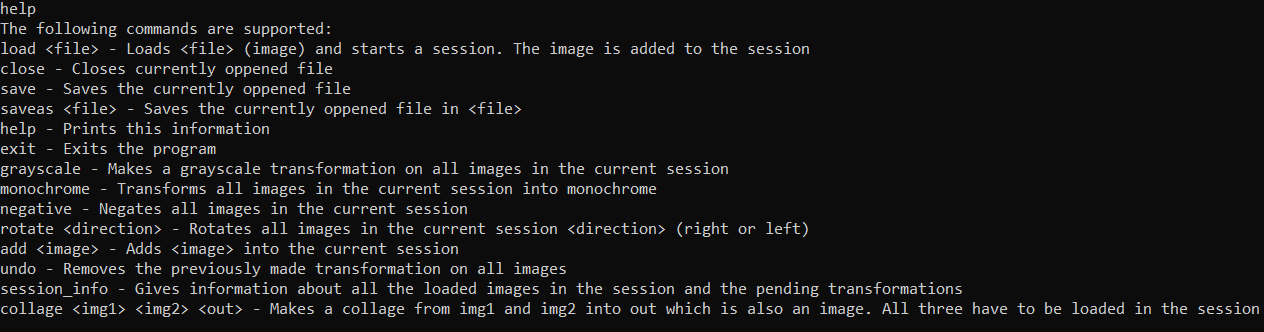
Прилагане на негатив върху снимки в сесията



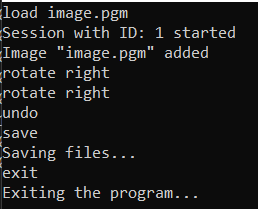
Прилагане на негатив върху снимки в сесията

Допълнителни примери за операциите:

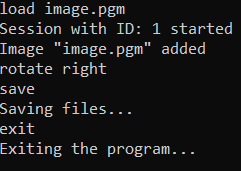
Функцията **help**

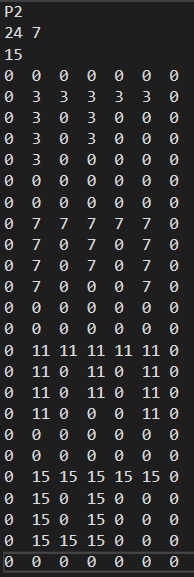


**Undo –** не ротира снимките

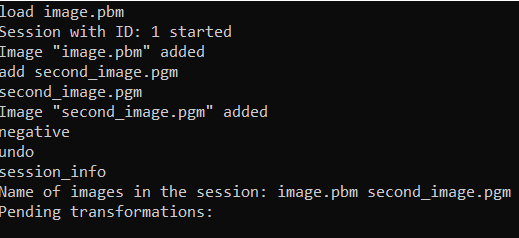


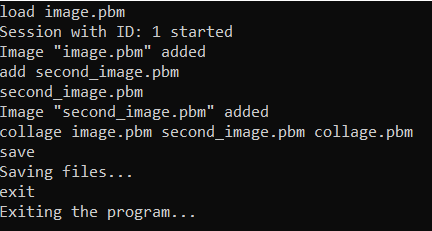
**Rotate**

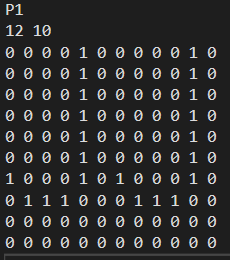




**Session\_info & Collage**





 **<= Резултатът**

Съответно получаваме колаж на двете снимки. Ако потребителя се опита да направи колаж от различни формати излиза грешка.

https://i.gyazo.com/46b2be2ed55d8b2c300e05f6639366b6.png

1. Заключение

Приложението има широк обхват от ползи и реализации. За базови операции, като тези описани в документацията е доста удобно и дава основа за развитие в областта на графичния софтуер. В бъдеще може определено да бъдат добавени повече формати за работа, например PNG или JPEG. С две думи мога да кажа, че информационната система е много интересна и приятна за употреба, както и полезна за усъвършенстването на графичен редактор.