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

Теми за проекти

курс Обектно-ориентирано програмиране
за специалност Софтуерно инженерство
Летен семестър 2018/2019 г.

Обща информация за проектите

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

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

По време на защитата се очаква да можете да отговорите на различни въпроси, като например: (1) каква архитектура сте избрали, (2) защо сте избрали именно нея, (3) дали сте обмислили други варианти и ако да — кои, (4) как точно работят различните части от вашия код и какво се случва на по-ниско ниво и др.

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

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

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

Начин на работа

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

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

Във всеки от проектите има посочен конкретен файлов формат, с който приложението ви трябва да работи. Това означава, че:
1. то трябва да може да чете произволен валиден файл от въпросния формат;
2. когато записва данните, то трябва да създава валидни файлове във въпросния формат.

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

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

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

Open

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

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

След като файлът бъде отворен и се прочете, той се затваря и приложението ви вече не трябва да работи с него, освен ако потребителят не поиска да запише обратно направените промени (вижте командата save по-долу), в който случай файлът трябва да се отвори наново. За целта трябва да изберете подходящо представяне на информацията от файла.

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

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

Close

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

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

Save

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

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

Save As

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

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

```
Successfully saved another file.xml
```

Exit

Излиза от програмата

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

Проект 1: Приложение за работа с електронни таблици

Представяне на данните

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

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

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

10	20	30	40
10		1000	
	10		

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

10, 20, 30, 40
 	
10,,1000,
,,,
,10
	10, 20 , 30 , 40
	
	10, , 1000,
	  , , ,
	  , 10

Типове данни в таблицата

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

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

```
123
-123
+123
```

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

```
123.456
-123.456
+123.456
```

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

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

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

Нужна функционалност

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

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

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

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

Извеждане на таблицата на екрана

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

Входен файл	Извеждане

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

Редактиране на клетки

Командата Edit трябва да позволява (с подходящи параметри) на потребителя да променя стойностите на отделните клетки. Това става като се укажат реда и колоната на

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

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

Формули

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

В дадена формула могат да участват единствено:

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

При сметките важат следните правила:

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

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

3. Ако в дадена формула участва празна клетка, тя се конвертира до нула. Това важи и за клетки, чиито координати надхвърлят размерите на таблицата.

4. Ако в дадена формула има грешка (например деление на нула), приложението ви не трябва да прекъсва своето изпълнение. Вместо това, когато то извежда таблицата на екрана, в съответната клетка се извежда ERROR, вместо получен резултат.

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

	Колона 1	**Колона 2**	**Колона 3**
Ред 1	10	Hello world!	123.56
Ред 2	123		

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

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

= R1C1 / R1C2	10 / 0	ERROR	

Проект 2: Работа със SVG файлове

В рамките на този проект трябва да се разработи приложение, което работи със файлове във Scalable Vector Graphics (SVG) формат. Приложението трябва да може да зарежда фигури от файла, да извършва върху тях дадени операции, след което да може да записва промените обратно на диска.

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

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

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

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

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

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

Операции

Print	Извежда на екрана всички фигури.
Create	Създава нова фигура.
Erase	Изтрива фигура
Translate	Транслира една или всички фигури. Ако потребителят не посочи конкретна фигура, тогава се транслират всички фигури; ако се посочи конкретна – променя се само тя.

Within	Извежда на екрана всички фигури, които изцяло се съдържат в даден регион. Потребителят може да укаже какъв да бъде регионът – кръг или правоъгълник

Примерен SVG файл figures.svg

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"
  "http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
<svg>
  <rect x="5" y="5" width="10" height="10" fill="green" />
  <circle cx="5" cy="5" r="10" fill="blue" />
  <rect x="100" y="60" width="10" height="10" fill="red" />
</svg>
```

Пример за работа на програмата

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

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

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

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

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

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

> erase 2
Erased a circle (2)
```

```
> erase 100
There is no figure number 100!

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

> translate vertical=10 horizontal=100
Translated all figures

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

> save
Successfully saved the changes to figures.svg

> exit
Exit
```

Проект 3: XML Parser

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

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

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

- Ако елементът има поле “id” във входния файл и стойността му е уникална за всички елементи от файла, да се ползва тази стойност.
- Ако елементът има поле “id” във входния файл, но стойността му не е уникална за всички елементи от файла, да се ползва тази стойност, но към нея да се конкатенира някакъв низ, който да допълни идентификатора до уникален низ.

(например, ако два елемента имат поле id=”1”, то единият да получи id=”1_1”, а другият -id=”1_2”)

- Ако елементът няма поле “id” във входния файл, да му се присъедини уникален идентификатор, генериран от програмата.

Програмата да дава следните възможности команди:

Open	Отваря и прочита валиден XML файл
Print	Извежда на екрана прочетената информация от XML файла (в рамките на посочените по-горе ограничения за поддържаната информация). Печатането да е XML коректно и да е “красиво”, т.е. да е форматирано визуално по подходящ начин (например, подчинените елементи да са по-навътре)
Select <id> <key>	Извежда стойност на атрибут по даден идентификатор на елемента и ключ на атрибута
Set <id> <key> <value>	Присвояване на стойност на атрибут
Children <id>	Списък с атрибути на вложените елементи
Child <id> <n>	Достъп до n-тия наследник на елемент
Text <id>	Достъп да текста на елемент
Delete <id> <key>	Изтриване на атрибут на елемент по ключ
Newchild <id>	Добавяне на НОВ наследник на елемент. Новият елемент няма никакви атрибути, освен идентификатор
XPath <id> <XPath>	операции за изпълнение на прости XPath 2.0 заявки към даден елемент, която връща списък от XML елементи

Минимални изисквания за поддържаните XPath заявки

Примерите по-долу са върху следния прост XML низ:

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

```
            <name>Ivan Petrov</name>
            <address>Bulgaria</address>
        </person>
</people>
```

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

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

Бонуси:

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

Проект 4: Недетерминиран краен автомат

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

Програмата да поддържа следните командит:

Open	Прочита от файл автомат, сериализиран по разработен от Вас формат. Всеки прочетен автомат да получава уникален идентификатор
List	Списък с идентификаторите на всички прочетени автомати
Print <id>	Извежда информация за всички преходи в автомата
Save <id> <filename>	Записва автомат във файл
Empty <id>	Проверява дали езикът на автомата е празен

Deterministic <id>	Проверява дали автомат е детерминиран
Recognize <id> <word>	Проверява дали дадена дума е в езика на автомата
Union <id1> <id2>	Намира обединението на два автомата и създава нов автомат. Отпечатва идентификатора на новия автомат.
Concat <id1> <id2>	Намира конкатенацията на два автомата и създава нов автомат. Отпечатва идентификатора на новия автомат.
Un <id>	Намира позитивна обвивка на автомат и създава нов автомат. Отпечатва идентификатора на новия автомат.
Reg <regex>	Създава нов автомат по даден регулярен израз (теорема на Клини). Отпечатва идентификатора на новия автомат.

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

Проект 5: Контекстно-свободна граматика

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

Програмата да поддържа следните команди:

Open	Прочита от файл граматика, сериализирана по разработен от Вас формат. Всяка прочетена граматика да получава уникален идентификатор.
List	Списък с идентификаторите на всички прочетени граматики
Print <id>	Извежда граматиката в подходящ формат. За всяко правило да се отпечата пореден номер.
Save <id> <filename>	Записва граматиката във файл
AddRule <id> <rule>	Добавя правила
RemoveRule <id> <n>	Премахване на правило по пореден номер
Union <id1> <id2>	Намира обединението на две граматики и създава нова

	граматика. Отпечатва идентификатора на новата граматика.
Concat <id1> <id2>	Намира сечението на две граматики и създава нова граматика. Отпечатва идентификатора на новата граматика.
Chomsky <id>	Проверява дали дадена граматика е в нормална форма на Чомски
CYK <id>	Проверява дали дадена дума е в езика на дадена граматика (CYK алгоритъм)
Iter <id>	Намира резултат от изпълнението на операцията “итерация” над две граматика и създава нова граматика. Отпечатва идентификатора на новата граматика.
Empty <id>	Проверява дали езика на дадена контекстно-свободна граматика е празен
Chomskify <id>	Преобразува граматика в нормална форма на Чомски. Отпечатва идентификатора на новата граматика.

Проект 6: Бази от данни

Да се реализира програма, поддържаща операции с прости бази от данни. Базите данни се състоят от серии от таблици, като всяка таблица е записана в собствен файл.

Поддържани типове данни

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

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

```
123
-123
+123
```

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

```
123.456
-123.456
+123.456
```

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

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

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

Програмата да позволява следните операции:

Load <file name>	Зарежда таблица от файл. Във файла е записана информация за типа на всяка колона. **Всяка таблица има име**. При опит за зареждане на таблица с име, което съвпада с името на някоя вече заредена таблица, системата да дава грешка.
Showtables	Показва списък с имената на всички заредени таблици
Describe <name>	Показва информация за типовете на колоните на дадена таблица
Print <name>	Показва всички редове от дадена таблица. Да се реализира диалогов режим, позволяващ съдържанието на таблицата да се преглежда по страници (такива, че се събират на един екран) със следните команди: следваща страница, предишна страница, изход.
Save <name> <file name>	Записва таблица във файл
Select <column-n> <value> <table name>	Извежда всички редове от таблицата, които съдържат стойността "value" в клетката с дадения пореден номер. Да се раализира извеждане по страници
AddColumn <table name> <column name>	Добавя нова колона (с най-голям номер) в дадена таблица. За всички съществуващи редове от таблицата, стойността на тази колона да е празна.

<column type>	
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 <tagret column n> да получи стойност <taget value>. Да се поддържа стойност NULL.
Delete <table name> <search column n> <search value>	Изтрива всички редове в таблицата, чиято колона <search column n> съдържа стойността <search column value>
Insert <table name> <column 1> … <column n>	Вмъква нов ред в таблицата със съответните стойности
InnerJoin <table 1> <column n1> <table 2> <column n2>	Извършва операцията Inner Join над две таблици спрямо колоните <column n1> в първата таблица и <column n2> във втората. Създава нова таблица и извежда идентификатора и.
Rename <old name> <new name>	Преименува таблица. Отпечатва грешка, ако новото име не е уникално.
Count <table name> <search column n> <search value>	Намира броя на редовете в таблицата, чиито колони съдържат дадената стойност
Aggregate <table name> <search column n> <search value> <target column n> <operation>	Извършва дадена операция върху стойностите от колоната <target column n> на всички редове, чиито колони с номер <search column n> съдържат стойността <search value>. Възможните операции са sum, product, maximum, minimum. Систамата да дава грешка, ако колоните не са числови.

За следващите два проекта важат други правила относно начин на работа

Вход

Приложението ви трябва да получава входа си от командния ред. Потребителят може да подава два вида аргументи – операции и пътища (пълни или относителни) към файловете, които трябва да се обработят. За да може двата вида аргументи да се различават, всички операции започват с два символа за тире (`--`).

Някои от операциите могат да имат аргументи, които определят начина им на работа. Ако операцията има аргументи, те се подават като след името ѝ се сложи символ за равенство и след него се подаде стойността на аргумента. Ето как може да изглежда един примерен вход:

```
prog.exe --comments --newlines=CRLF --tabs=tabs file.cpp C:\temp\1.cpp
```

Тук приложението ви получава три операции (`--comments`, `--newlines` и `--tabs`). Две от тях имат аргумент. Това са `--newlines`, която има за аргумент символния низ `CRLF` и `--tabs`, която има за аргумент символния низ `tabs`. `file.cpp` е относителното име на файл, който програмата трябва да обработи, а `C:\temp\1.cpp` – пълно име на друг файл.

Потребителят може да указва операциите и имената на файловете в какъвто пожелае ред. Например дадената по-долу команда трябва да се обработи идентично на предишната:

```
prog.exe file.cpp --comments --newlines=CRLF C:\temp\1.cpp --tabs=tabs
```

При това има две правила, които трябва да се спазват:

1. Файловете се обработват точно в реда, в който са подадени от командния ред. Тоест в случая `file.cpp` ще се обработи преди `1.cpp`.
2. Освен ако не е казано друго, за всеки отделен файл, операциите също се изпълняват точно в реда, в който са дадени.

Това означава, че в нашия пример последователността на операциите ще изглежда така:

1. Изпълнява се `--comments` върху `file.cpp`;
2. Изпълнява се `--newlines` върху `file.cpp`;

3. Изпълнява се `--tabs` върху `file.cpp`;
4. Изпълнява се `--comments` върху `1.cpp`;
5. Изпълнява се `--newlines` върху `1.cpp`;
6. Прилага се `--tabs` върху `1.cpp`;

Проект 7: Обработка на изходен код

В рамките на този проект трябва да се разработи приложение, което получава като вход един или повече файлове с изходен код на C/C++ и прилага върху тях серия от операции. Операциите,

които приложението трябва да изпълнява върху изходния код са описани по-долу.

При стартирането на приложението, потребителят избира кои операции да се извършват върху файловете и в какъв ред. Ако той избере две или повече операции, прилагането им върху файловете трябва да бъде последователно, т.е. докато не приключи прилагането на една операция върху даден фрагмент изходен код, не може да се премине към следващата. Някои от операциите могат да имат входни параметри, които определят начина им на работа.

Операции

Премахване на коментари от файл

Команда: `--comments`

Тази операция премахва всички коментари от даден фрагмент код. Операцията няма параметри. Тя премахва коментари от вид /* */ и //. Резултатът от обработката е нов фрагмент, от който са премахнати коментарите, а останалото съдържание е същото като в оригинала.

Нормализиране на нови редове

Команда: `--newlines=<format>`

Новите редове в един фрагмент могат да бъдат в различен формат (например някои редове могат да завършват с `\n`, а други с `\r\n`). Операцията трябва да коригира редовете във фрагмента, като ги конвертира до посочен от потребителя формат.

Операцията има един аргумент, който може да има една от следните стойности:

- CRLF – всички нови редове стават \r\n (т.е. \n се подменя с \r\n)
- LF – всички нови редове стават \n

Нормализиране на отстъпа

Команда: `--indentation=<format>`

В даден фрагмент код отстъпите могат да бъдат въведени със символ за табулация (\t) или с четири символа за интервал. Възможно е във фрагмента тези два стила да са смесени (т.е. в един ред да има табулации, в друг – интервали, в трети – и табулации, и интервали и т.н.). Операцията за нормализиране на отстъпите получава един аргумент, който указва кой от двата стила предпочита потребителят – табулации (tabs) или интервали (spaces). Резултатът е нов фрагмент, в който всички отстъпи са форматирани в желания формат. Например ако потребителят предпочита табулации,

операцията трябва да подмени всички срещания на четири интервала със символ за табулация.

Подреждане на код

Команда: `--format`

Тази операция няма аргументи. Тя получава фрагмент код и трябва да оправи форматирането в него там, където е нужно. Това трябва да стане по следния начин:

1. Всяка инструкция (statement) трябва да започва на отделен ред.

ПРЕДИ:	СЛЕД:
`int x, y; if( 1 > 2) return;`	`int x, y;` `if( 1 > 2) return;`

2. Вложените блокове и телата на `if`, `for`, `while` и т.н. трябва да бъдат на нов ред и с един отстъп навътре. Например:

ПРЕДИ:	СЛЕД:

`if(1 > 2) return; else`
`  {`
`  int x, y;`
`int z;`
`  }`
	`if(1 > 2)`
	`  return;`
	`else`
	`{`
	`  int x, y;`
	`  int z;`
	`}`

Оцветяване на код

Команда: `--html`

Операцията получава фрагмент код и го форматира като HTML. Операцията няма аргументи.

Резултатът трябва да бъде валиден HTML код, в който са маркирани:

- Коментарите
- Запазените думи в езика
- Символните низове (стрингове)
- Препроцесорните директиви
- Числовите литерали (напр. 1, 5.5, .9, 1e3)
- Символните литерали (напр. ‘c’, ‘\t’)

Самите изберете какво да бъде маркирането. Например запазените думи може да са **сини и с получер шрифт (bold)**, коментарите да са в *зелен курсив*, числовите литерали да са оранжеви и т.н.

ВАЖНО: Това е единствената команда, за която не важи правилото за последователно прилагане. Независимо къде е указана тази команда, тя трябва да се изпълни само веднъж и то накрая, след като са изпълнени останалите операции върху дадения файл.

Вход

Важат същите правила, които са описани в общата информация за проектите. Потребителят може да подава един или повече входни файлове.

Изход

Ако потребителят избере опция да генерира HTML файл, за всеки входен файл, програмата трябва да генерира нов файл със същото име и разширение `.html`. Например

```
prog.exe --html --indentation=tabs file.cpp C:\temp\1.cpp
```

трябва да генерира два нови файла: `file.html` и `C:\temp\1.html`

Ако потребителят не е избрал опция за генериране на HTML файл, старото съдържание на файловете трябва да се запише във файлове с разширение .old, а съществуващите файлове да се променят. Например

```
prog.exe --tabs=tabs file.cpp C:\temp\1.cpp
```

трябва да генерира два файла `file.cpp.old` и `C:\temp\1.cpp.old`, в които ще запази старото съдържание, а във файловете `file.cpp` и `1.cpp` ще се запише резултатът от обработката.

Проект 8: Обработка на изображения PPM

В рамките на този проект трябва да се разработи приложение, което получава като вход графичен файл и прилага върху него някакви операции.

Приложението ви трябва да може да работи с различни видове файлове. При стартирането му потребителят подава път (пълен или относителен) към един файл, а приложението трябва само да определи какъв е неговият формат, да извърши операциите и да генерира нов файл в същия формат (например ако входният файл е в PPM формат, изходният също трябва да бъде PPM файл.

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).

Операциите, които трябва да се поддържат са описани по-долу

Конвертиране до чернобяло изображение (grayscale)

Команда: `--grayscale`

Ако на приложението бъде подаден цветен файл, по него трябва да се генерира нов черно-бял файл, който да се запише в същата директория като входния файл. Новият файл трябва да бъде със същото име и суфикс `_grayscale`.

Ако се подаде чернобял или монохромен файл, той не се променя и остава същият. Забележете, 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). В такъв случай програмата ви отново не трябва да генерира нищо. Например

```
> prog.exe --grayscale image01.ppm
```

ще генерира нов файл с име `image01_grayscale.ppm`, ако файлът е цветен. Ако той е чернобял, програмата няма да генерира нищо.

Конвертиране до монохромно изображение (monochrome)

Команда: `--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). Новият

файл трябва да бъде със същото име и суфикс `_monochrome`. Отново, ако входният файл е монохромен, програмата не прави нищо.

Изчисляване на хистограма

Команда: `--histogram=<channel>`

Ако потребителят подаде тази команда, той подава и коя хистограма иска да се генерира – за зелените пиксели (green), за червените (red) или за сините (blue). След това програмата генерира нов цветен файл съдържащ хистограмата (сами изберете формата на този файл, например може да използвате PPM). Новият файл е със същото име като оригиналния, но със суфикс `_histogram_<channel>`. Например:

```
> prog.exe --histogram=red --histogram=green KonPieBoza.ppm
```

ще генерира два файла с имена `KonPieBoza_histogram_red.ppm` и `KonPieBoza_histogram_green.ppm` съответно за хистограми за червения и зеления канал.

Хистограмата трябва да бъде графика с височина 100 и широчина 255 (т.е. 255 колони с височина по 100 пиксела). Накратко правилото за генериране на хистограмата е:

В хистограма за канал *X*, колоната *W* ще има височина *H*, когато *H%* от пикселите в изображението имат стойност *W* в съответния си компонент *X*.

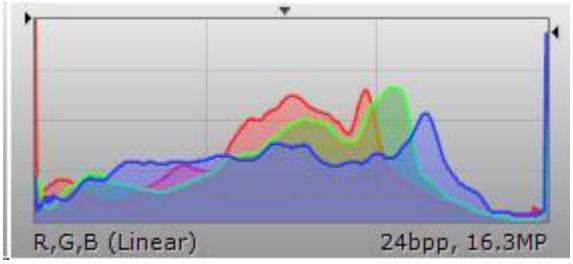
Казано по-просто: Ако генерираме хистограма за зеления канал и в изображението

- 10% от пикселите имат стойност 0 за зелено;
- 50% от пикселите имат стойност 153 за зелено;
- 40% имат стойност 255 за зелено,

то хистограмата ще има само три колони – една на позиция 0 с височина 10, една на позиция 153 с височина 50 и една на позиция 255 с височина 40. Тогава хистограмата може да изглежда така:

Ако се генерира хистограма за зеления канал, тя трябва да бъде нарисувана в зелено, а тези за синия и червения – съответно в синьо и червено.

Допълнителни точки ще бъдат дадени, ако графиката на хистограмата има надписи и е оформена по-красиво и/или към програмата се добавят нови опции, като например изобразяване на трите хистограми на една диаграма, например[1]:



Вход

Важат същите правила, които са описани в общата информация за проектите. Потребителят може да подава един или повече входни файлове, като всички се обработват по горе-описаните правила. Тъй като операциите са независими една от друга, можете да решите в какъв ред да ги изпълните. Забележете, че при обработката оригиналните файлове **не трябва** да се променят, а да се генерират нови файлове.

[1] Показаната диаграма е генерирана с FastPictureViewer 1.9 (http://www.fastpictureviewer.com/)

Изход

Новогенерираните файлове трябва да бъдат в същите директории, като файловете, които са ги породили. Например ако обработваме `C:\temp\1.ppm`, новите файлове също трябва да бъдат в директория `C:\temp`.