КУРСОВИ ПРОЕКТИ ПО ОБЕКТНО-ОРИЕНТИРАНО ПРОГРАМИРАНЕ

Факултет по математика и информатика на СУ „Св. Климент Охридски", Специалност „Информатика“, 1 курс

Летен семестър 2014/2015

Обща информация за проектите

Проектите се оценяват по редица от критерии, всеки от които носи точки. Общата сума от точките надвишава необходимия брой за оценка отличен (т.е. възможно е един проект да бъде оценен с отличен дори и да не събере максимално възможния брой точки). Някои от основните критерии за оценка 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.

Използване на чужд код

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 при предаването и защитата на проекта, като посочите коя част от проекта сте разработили самостоятелно. Това означава, че:

1. Използваните наготово парчета код трябва да са маркирани ясно, като поставите коментари на подходящи места.
2. По време на защитата трябва да посочите кои части сте разработили самостоятелно и кои са взети от други източници.

Когато в даден проект се използва чужд код се оценяват и (1) способността ви за внедряване на този код във вашето решение (напр. в случаите, когато се използва външна библиотека) и (2) дали добре разбирате какво прави чуждият код.

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`;

1: 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.

2: Обработка на изображения

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*.

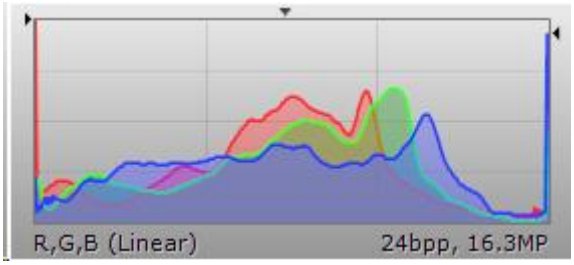
Казано по-просто: Ако генерираме хистограма за зеления канал и в изображението

- 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

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

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.

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`.