Технологично училище "Електронни системи"
Технически университет – София

Обектно-ориентирано програмиране

Домашна работа: Реализация на SVG графика Tutle Graphics

Любомир Чорбаджиев
lchorbadjiev@elsys-bg.org

Емил Гоцев
egotsev@gmail.com

7 февруари 2018 г.

1 Условие на задачата

Целта на задачата е да се направи SVG имплементация на класа `Turtle`.

Като основа за имплементацията трябва да използвате класовете `Turtle` и `PSTurtle`, както и помощните класове `Point` и `Color` дефинирани в

- https://github.com/elsys/oop2017-2018/tree/master/homeworks/11-turtle-graphics
- https://github.com/elsys/oop2017-2018/blob/master/homeworks/11-turtle-graphics/turtle.hh
- https://github.com/elsys/oop2017-2018/blob/master/homeworks/11-turtle-graphics/turtle.cc

Тези класове са подобни на разработени в часовете

- https://github.com/elsys/oop2017-2018/tree/master/11a/2018-01-16-turtle
- https://github.com/elsys/oop2017-2018/tree/master/11b/2018-01-18-turtle
- https://github.com/elsys/oop2017-2018/tree/master/11a/2018-01-25-maze

и описани накратко в

- https://github.com/elsys/oop2017-2018/blob/master/practice/turtle.md

1.1 Реализация на class `SVGTurtle` (50 точки)

Дефинирайте клас `SVGTurtle`, който наследява базовия клас `Turtle` дефиниран в примерите.

Дефинираният от вас клас трябва да поддържа следния интерфейс:

```
1
2 class SVGTurtle: public Turtle {
3   std::ostream& out_;
4
5 public:
6   SVGTurtle(double width, double height,
7             std::ostream& out);
8   virtual ~SVGTurtle();
9
10  virtual void setup();
11
12  virtual Turtle& moveto(const Point& p);
13 };
```

При дефинирането на класа `SVGTurtle` не трябва да променяте имплементацията на базовия клас и на вече дефинирания наследник `PSTurtle`.

- Конструкторът на класа получава като аргументи ширината `width` и височината `height` на полето за рисуване, както и поток за изход `out`, в който да записва генерираната `HTML` страница.

  ```
  SVGTurtle(double width, double height,
            std::ostream& out);
  ```

- Методът `void setup()` трябва да запише в потока `out_` заглавната част на `HTML` файла, съдържащ `SVG` фигурата. Например:

```
<html>
<body>

<h1>SVG Turtle Graphics</h1>

<svg width="100" height="100">
```

- Деструкторът на класа `~SVGTurtle` трябва да запише в потока `out_` затварящите тагове на `SVG` файла. Например:

```
</svg>

</body>
</html>
```

- Методът

```
virtual Turtle& moveto(const Point& p);
```

  трябва да премести "костенурката" от текущата ѝ позиция до позиция `const Point& p` която става новата текуща позиция на "костенурката". Ако "костенурката" е със спуснат писец, то тази операция трябва да създаде `SVG line` елемент от първоначалната до новата позиция на "костенурката". Например:

```
...
  <line x1="50" y1="50" x2="60" y2="60"
        style="stroke:rgb(255,0,0);stroke-width:2" />
...
```

  Атрибутите на създадената линия трябва да съответстват на зададените цвят и дебелина на линията на "костенурката".

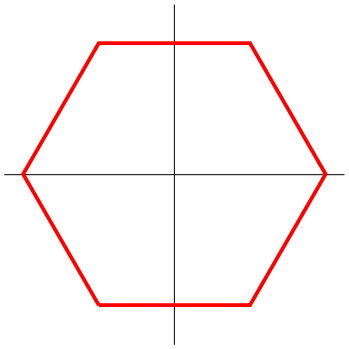
Дефинираният от вас клас `SVGTurtle` трябва да бъде в отделен файл (файлове). Предоставените начални файлове `turtle.hh` и `turtle.cc` не трябва да се променят. Всяка промяна на някой от първоначално предоставените файловете `turtle.hh` или `turtle.cc` може да доведе до намаляване на получените от вас точки.

1.2 Рисуване на фигури (20 точки)

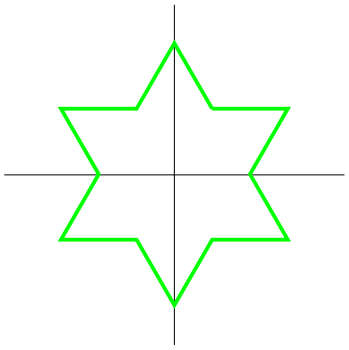
1.2.1 Червен правилен шестоъгълник

Като използвате разработения от вас клас `SVGTurtle` и съществуващия клас `PSTurtle` напишете програма, която рисува правилен шестоъгълник в канвас с размери 1000×1000.

Програмата трябва да рисува координатните оси с център в точка с координати $(500, 500)$. Страната на шестоъгълника трябва да е с дължина 400. Центърът на шестоъгълника трябва да съвпада с центъра на координатната система - виж фигура 1.



Фигура 1: Червен правилен шестоъгълник



Фигура 2: Зелена шестоъгълна звезда

Изпълнимият файл трябва да се казва `hexagon` и трябва да приема един аргумент от командния ред, който определя в какъв формат се рисува. Допустимите стойности на аргумента са:

- `eps` – програмата трябва да нарисува специфицирания шестоъгълник в `EPS` формат като използва класа `PSTurtle`
- `svg` – програмата трябва да нарисува специфицирания шестоъгълник в `SVG` формат като използва класа `SVGTurtle`

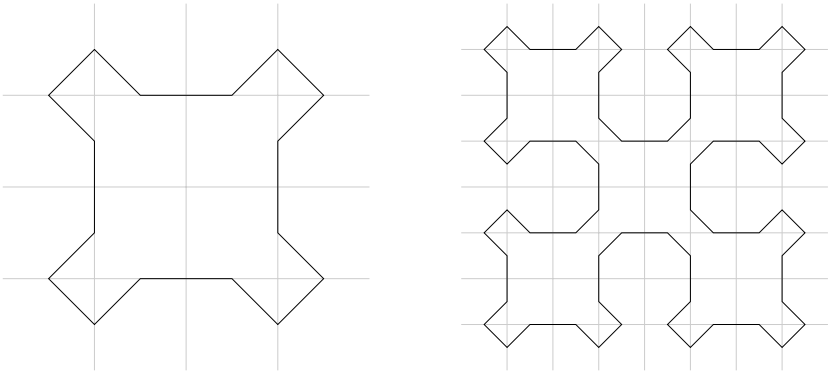
Примерно извикване на програмата може да бъде:

```
./hexagon eps > hexagon.eps
./hexagon svg > hexagon.html
```

1.2.2 Зелена шестоъгълна звезда

Като използвате разработения клас `SVGTurtle` и съществуващия клас `PSTurtle` напишете програма, която рисува зелена шестоъгълна звезда в канвас с размери 1000×1000.

Програмата трябва да рисува координатните оси с център в точка с координати $(500, 500)$. Страната на звездата трябва да бъде 200. Центърът



(а) Крива не Серпински S_1 от ред 1 (б) Крива на Серпински S_2 от ред 2

Фигура 3: Криви на Серпински от 1 и 2 ред. (Очаква се програмата ви да рисува само кривата на Серпински без координатната решетка показана на фигурата.)

на звездата трябва да съвпада с центъра на координатната система - виж фигура 2.

Изпълнимият файл трябва да се казва `star` и трябва да приема един аргумент от командния ред, който определя в какъв формат се рисува. Допустимите стойности на аргумента са:

- `eps` – програмата трябва да нарисува специфицираната звезда в `EPS` формат като използва класа `PSTurtle`
- `svg` – програмата трябва да нарисува специфицираната звезда в `SVG` формат като използва класа `SVGTurtle`

Примерно извикване на програмата може да бъде:

```
./star eps > star.eps
./star svg > star.html
```

1.3 Крива на Серпински (30 точки)

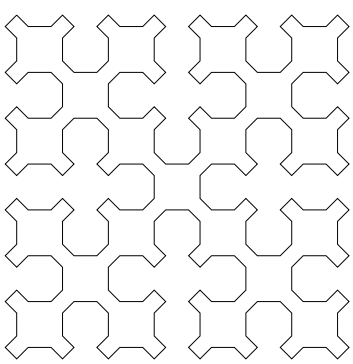
Като използвате разработения клас `SVGTurtle` и съществуващия клас `PSTurtle` напишете програма, която рисува криви на Серпински от различен ред.

Кривите на Серпински са рекурсивни криви подобни на имплементираните в часовете криви на Хилберт.

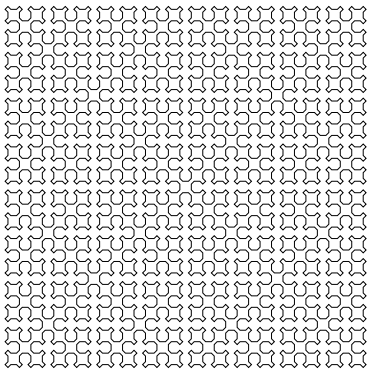
Програмата трябва да работи върху канвас с размер 2048×2048. Като параметър на програмата трябва да се предаде кой ред крива на Серпински трябва да бъде нарисувана.

Програмата трябва да изчисли всички необходими параметри за рисуването на съответната крива на Серпински и да я разположи върху канваса, както е показано на фигурите 3, 4 и 5.

Изпълнимият файл трябва да се казва `sierpinski` и трябва да приема два аргумент от командния ред. Първият аргумент определя в какъв формат се рисува. Допустимите стойности на аргумента са:



(а) Крива не Серпински S_3 от ред 3



(б) Крива на Серпински S_5 от ред 5

Фигура 4: Криви на Серпински от 3 и 5 ред

- `eps` – програмата трябва да нарисува специфицираната звезда в `EPS` формат като използва класа `PSTurtle`
- `svg` – програмата трябва да нарисува специфицираната звезда в `SVG` формат като използва класа `SVGTurtle`

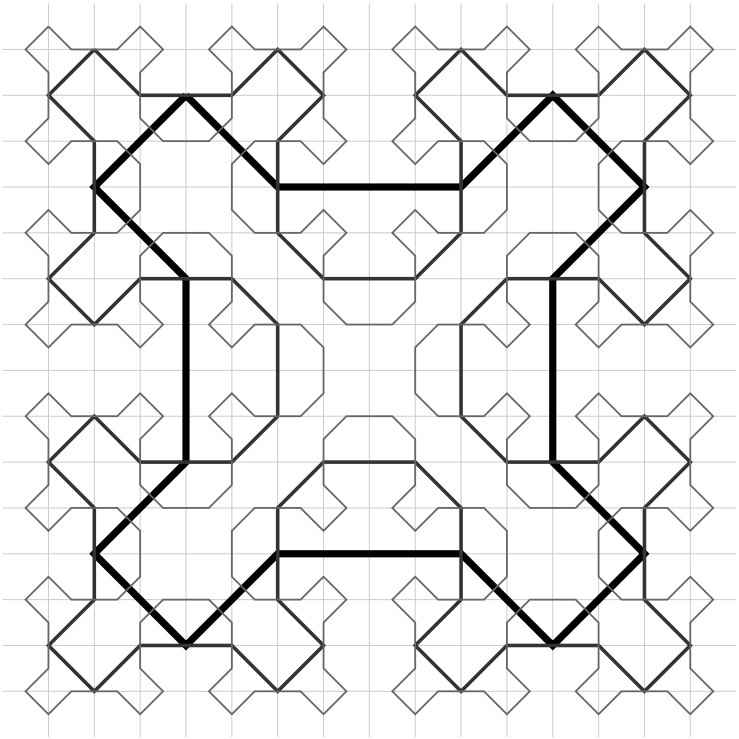
Вторият аргумент определя от какъв ред трябва да бъде нарисуваната крива на Серпински. Този аргумент трябва да бъде цяло число с допустими стойности $n = 1, 2, 3, 4, 5, 6, 7$.

Примерно извикване на програмата може да бъде следното:

```
./sierpinski eps 1 > sierpinski_1.eps
./sierpinski svg 1 > sierpinski_1.svg
./sierpinski eps 3 > sierpinski_3.eps
./sierpinski svg 3 > sierpinski_3.svg
```

2 Изисквания към решението и оценяване

1. Решението на задачата трябва да бъде написано на езика `C++` съгласно ISO/IEC 14882:1998, или ISO/IEC 14882:2011 или ISO/IEC 14882:2017.
2. Домашната работа се предава като `ZIP` архив именуван `turtle.zip`, който трябва да съдържа
   - всички необходими изходни файлове, включително предоставените начални файлове `turtle.hh` и `turtle.cc`;
   - `Makefile` който компилира всички необходими файлове и създава всички необходими изпълними файлове - `hexagon`, `star` и `sierpinski`;
   - предаденият архив не трябва да съдържа изпълними файлове, файлове с обектен код или каквито и да е било други файлове резултат от компилацията; наличието на подобни файлове в архива ще доведе до намаляване на резултата ви с до 30 точки.



Фигура 5: Криви на Сепрински от ред 1, 2 и 3

3. Правилата за оценяване са следните. Приемаме, че напълно коректна и написана спрямо изискванията програма получава максималния брой точки — 100% или 100 точки. Ако в решението има пропуски, максималният брой точки ще бъде намален съгласно правилата описани по-долу.

4. При проверка на решението програмата ви ще бъде компилирана и тествана по следния начин:

```
make
./star eps > star.eps
./sierpinski eps 3 > sierpinski.eps
./sierpinski svg 2 > sierpinski.svg
```

   Предходната процедура ще бъде изпълнена няколко пъти с различни входни данни за да се провери дали вашата програма работи коректно.

5. Реализацията на програмата трябва да спазва точно изискванията. Всяко отклонение от изискванията ще доведе до получаване на 0 точки за съответната част от условието.

6. Работи, които са предадени по-късно от обявеното (или не са предадени), ще бъдат оценени с 0 точки.

7. Програмата ви трябва да съдържа достатъчно коментари. Оценката

на решения без коментари или с недостатъчно и/или мъгляви коментари може да доведе до намаляване на оценката ви с 30 точки.

8. Всеки файл от решението трябва да започва със следният коментар:

```
//--------------------------------------------
// NAME: Ivan Ivanov
// CLASS: XIa
// NUMBER: 13
// PROBLEM: #1
// FILE NAME: xxxxxx.yyy.zzz (unix file name)
// FILE PURPOSE:
//  няколко реда, които описват накратко
//  предназначението на файла
//  ...
//--------------------------------------------
```

   Всяка нетривиална функция (функция с дължина повече от 7 реда) във вашата програма трябва да включва кратко описание в следния формат:

```
//--------------------------------------------
// FUNCTION: xxyyzz (име на функцията)
//  предназначение на функцията
// PARAMETERS:
//  списък с параметрите на функцията
//  и тяхното значение
//--------------------------------------------
```

9. Лош стил на програмиране и липсващи заглавни коментари ще ви костват 30%.

10. Програми, които не се компилират, получават 0 точки. Под „не се компилират“ се има предвид произволна причина, която може да доведе до неуспешна компилация, включително липсващи файлове, неправилни имена на файлове, синтактични грешки, неправилен или липсващ `Makefile`, и т.н. Обърнете внимание, че в `UNIX` имената на файловете са “case sensitive”.

11. Програми, които се компилират, но не работят, не могат да получат повече от 50%. Под „компилира се, но не работи“ се има предвид, че вие сте се опитали да решите проблема до известна степен, но не сте успели да направите пълно решение. Често срещан проблем, който спада към този случай, е че вашият Makefile генерира изпълним файл, но той е именуван с име, различно от очакваното.

12. Безсмислени или мъгляви програми ще бъдат оценявани с 0 точки, независимо че се компилират.

13. Програми, които дават неправилни или непълни резултати, или програми, в които изходът и/или форматирането се различава от изискванията, ще получат не повече от 70%.

14. Всички наказателни точки се сумират. Например, ако вашата програма няма задължителните коментари в началото на файлове и функциите, се отнемат 30%, ако няма достатъчно коментари, се отнемат още 30%, компилира се, но не работи правилно — още 30%, то тогава резултатът ще бъде: $80 * (100 - 30 - 30 - 30)\% = 100 * 10\% = 10$ точки

15. Работете самостоятелно. Групи от работи, които имат твърде много прилики една с друга, ще бъдат оценявани с 0 точки.