Софийски университет „Св. Климент Охридски“

Факултет по математика и информатика

КУРСОВ ПРОЕКТ ПО ОБЕКТНО-ОРИЕНТИРАНО ПРОГРАМИРАНЕ

летен семестър 2020/2021г.

Проект 2 – Работа със SVG файлове

Изготвил:Атанас Живков Атанасов

Специалност: Информационни системи

Факултетен номер: 72028

Група: 1

Ръководител на курс: Калин Георгиев

Май 2021

гр.София

Съдържание

1.1 *Описание и идея на проекта*

Основната идея на проекта е 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.

Имайки предвид безбройните фигури и 3D обекти, които приложението БИ МОГЛО да поддържа, ще се ограничa с 3 основни фигури: Rectangle(Правоъгълник), Circle(Кръг), Line(Линия)!

Също така, координатната система, използвана за проектирането и функционалносста на фигурите е стандартната: положителната полуос X сочи надясно, а положителната полуос Y сочи надолу.

2.1. *Основни дефиниции, концепции и алгоритми*

Проектът има две остовни части: Base структурата - Shape и Mega класът SVG.

Shape е основата на трите фигури, които форматираме и които съхраняваме във файла. Форматирайки го като Pure virtual (изцяло виртуален) клас ни позволява да постигнем по-добра абстракция и да спазим основните ООП принципи. Shape бива наследяван от Rectangle, Circle и Line, които са и фигурите, с които работим.

SVG е Mega класът, който също представлява и “логиката” на проекта. Той наследява помощния клас Menu, в който биват съхранени командите, използвани в терминала за работа с файла. Важните алгоритми, описани в SVG са within и двата метода за работа с файл: saveToFile и loadFromFile.

Within функцията е Pure virtual за shape класът, за да може всеки наследяващ клас да я модифицира(overwrite)спрямо параметрите си. В SVG функцията се (overload-ва) за да се изпълнява когато се подават съответно Rectangle& и Circle&(по условие).

```
}

Vector<Shape*> SVG::within(Rectangle& newShape)
{
    Vector<Shape*> figures_within;
    for (size_t i = 0; i < figures.getSize(); i++)
    {
        if (figures[i]->within(newShape.getX(),newShape.getY(),newShape.getWidth(),newShape.getHeight()))
        {
            figures_within.push_back(figures[i]);
        }
    }
    return figures_within;
}
Vector<Shape*> SVG::within(Circle& newShape)
{
    Vector<Shape*> figures_within;
    for (size_t i = 0; i < figures.getSize(); i++)
    {
        if (figures[i]->within(newShape.getX(),newShape.getY(),newShape.getRadius()))
        {
            figures_within.push_back(figures[i]);
        }
    }
    return figures_within;
}
```

Функцията е дефинирана във различните класове, приемайки Int стойности, за да се избегне така нареченият Double Matching Problem

2.2 *Подходи, методи за решаване на поставените проблеми*

Имайки предвид, че основната функционалност на програмата се изпълнява чрез подадени команди, в класът SVG имплементираме функцията start, която се базира на условната конструкция -(switch - case), като така извършва валидация на подадените команди и ни “изпраща” в съответния(от 10 възможни) case-ове. Съответният case вика нужните му функции и идпълнява зададената (по условие) функционалност.

```
Menu::Menu()
{
    menu.push_back("Open");//0
    menu.push_back("Close");//1
    menu.push_back("Save");//2
    menu.push_back("Saveas");//3
    menu.push_back("Print");//4
    menu.push_back("Within");//5
    menu.push_back("Erase");//6
    menu.push_back("Create");//7
    menu.push_back("Translate");//8
    menu.push_back("Help");//9
    menu.push_back("Exit");//10
}
```

```
class SVG : public Menu
{
    private:
    Vector<Shape*> figures;//the vector that holds all the figures.
    Vector<String> commands;
    char* filename;

    public:
    SVG();
    ~SVG();
    void addFigure(Shape* figure);
    Shape* createFigure ();
    void saveToFile();
    void loadFromFile();
    void print();

    Vector<Shape*> within(Rectangle& newShape);
    Vector<Shape*> within(Circle& newShape);
    void translate();

    void start();
};
```

3.1. *Обща архитектура – ООП дизайн*

Основната структура - Shape позволява изключителна гъвкавост, при добавяне на нови фигури, тъй като е “изградена”, така че да позволява overwrite на virtual/pure virtual функциите, както и да спести допълнителното писане на setters/getters (което си е и ООП практика за base class-а).

Липсата на готовите STL контейнери std::vector и std::string значително ускорява програмата, и подобрява нейната производителност. Отново, липсата на готови библиотеки позволява по-гъвкава редакция по кода, в случай на правене на промени, които не присъстват в стандартните библиотеки.

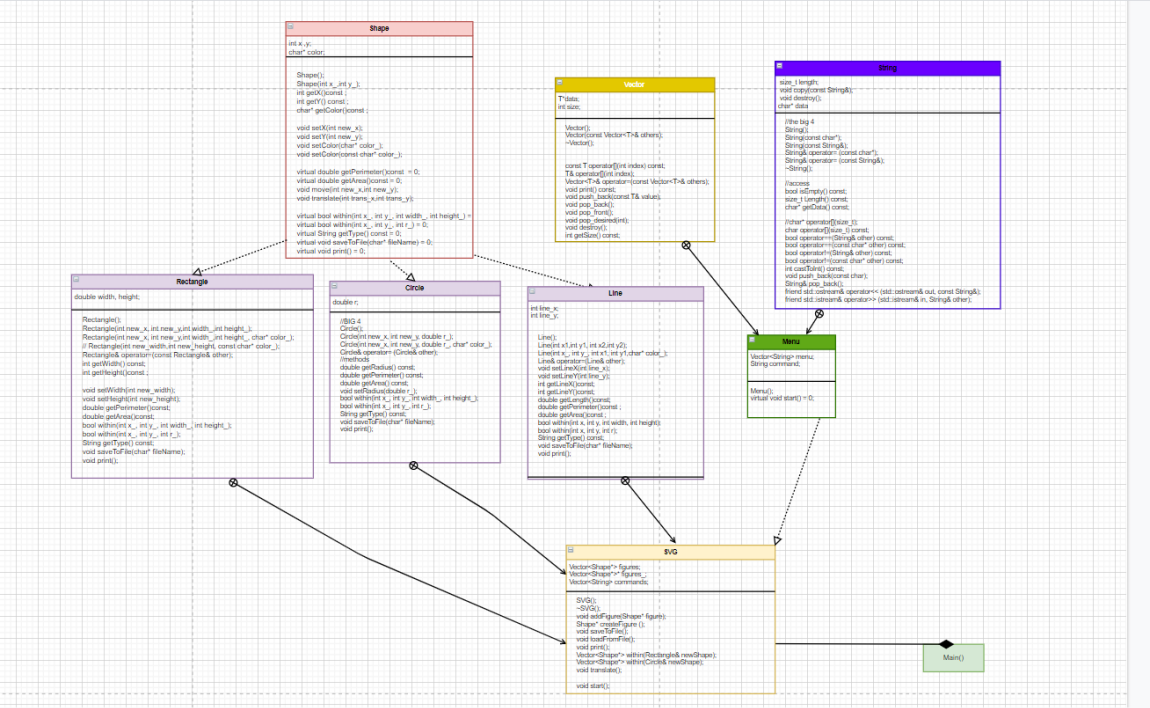
Пример: pop_desired -> class Vector

```
}
template<typename T>
void Vector<T>::pop_desired(int position)
{
    T* temp = new T[size - 1];
    for (size_t i = 0; i < position; i++)
    {
        temp[i] = data[i];
    }
    size--;
    for (size_t i = position; i < size ; i++)
    {
        temp[i] = data[i + 1];
    }
    delete[] data;
    data = temp;

}
template<typename T>
```

Премахва елемент, според подаден от нас индекс(Имплементирана е в командата Erase!).

3.2. *UML диаграма*



4.1. *Реализация на класове*

Така наречените по-важни „моменти“ в проекта са разделени на отделни класове , като по този начин се улеснява изключително много работата не само в процеса на разработка на проекта, но и в процеса на неговото подобряване. Това разделение подобрява също така четимостта на кода.

Проектът съдържа следните класове:

Първият клас, който всъщност е структура е Shape, който е основата на всяка една от използвните фигури (общо 3 на брой).

```
struct Shape
{
    int x; // indicates RIGHT
    int y; //indicates DOWN
    char* color;

    Shape();
    Shape(int x_,int y_);
    int getX()const ;
    int getY() const ;
    char* getColor()const ;


    void setX(int new_x);
    void setY(int new_y);
    void setColor(char* color_);
    void setColor(const char* color_);

    virtual double getPerimeter()const  = 0;
    virtual double getArea()const = 0;

    void move(int new_x,int new_y);// moves the default shape - coord 0,0 to the actual place
    void translate(int trans_x,int trans_y);// it's the translation of the newly created figure

    virtual bool within(int x_, int y_, int width_, int height_) = 0;
    virtual bool within(int x_, int y_, int r_) = 0;
    virtual String getType() const = 0;
    virtual void saveToFile(char* fileName) = 0;
    virtual void print() = 0;

};
```

Shape приема началните координати (x ,y)на нашите фигури и осигурява началното конструиране на фигурата. Всяка фигура бива формирана по различен начин:

Rectangle започва от долният ляв ъгъл.

Circle започва от средата на кръга.

За Line, Shape се грижи за конструирането на първата(от двете) точка, нужна за образуването на линия.

Следващите класове (Rectangle, Circle и Line) са наследниците на Shape, като (бивайки включени в SVG по-късно) изпълняват и четирите основни пронципа на ООП:

Капсулация: Не може да се достига директно до private данните на класовете. За тази цел, имаме “getters” и “setters”.

Пример: В класът Line

```
double Line::getPerimeter()const {return getLength();}
double Line::getArea()const {return 0;}
```

Наследяване:

Rectangle/Circle/Line наследяват Shape публично, защото нямаме нужда от private или protected наследяване, тъй като нивото на ограничаване на данните не е толкова високо, а и имаме доста методи, които изискват по - голяма достъпност.

Абстракция:

Възможността да “изграждаме” обекти, използвайки Shape индиректно е изключително полезна за нас.Тъй като е невъзможно директно да изградим обект от клас Shape, при подаване или връщане на Shape - заменяме Shape с Shape*, за да “посочим” че искаме да използваме Shape.

Другият основен клас за програмата е Mega класът - SVG

```
8
9
10  class SVG : public Menu
11  {
12      private:
13      Vector<Shape*> figures;//the vector that holds all the figures.
14      Vector<String> commands;
15      char* filename;
16
17      public:
18      SVG();
19      ~SVG();
20      void addFigure(Shape* figure);
21      Shape* createFigure ();
22      void saveToFile();
23      void loadFromFile();
24      void print();
25
26      Vector<Shape*> within(Rectangle& newShape);
27      Vector<Shape*> within(Circle& newShape);
28      void translate();
29
30      void start();
31  };
32
33
```

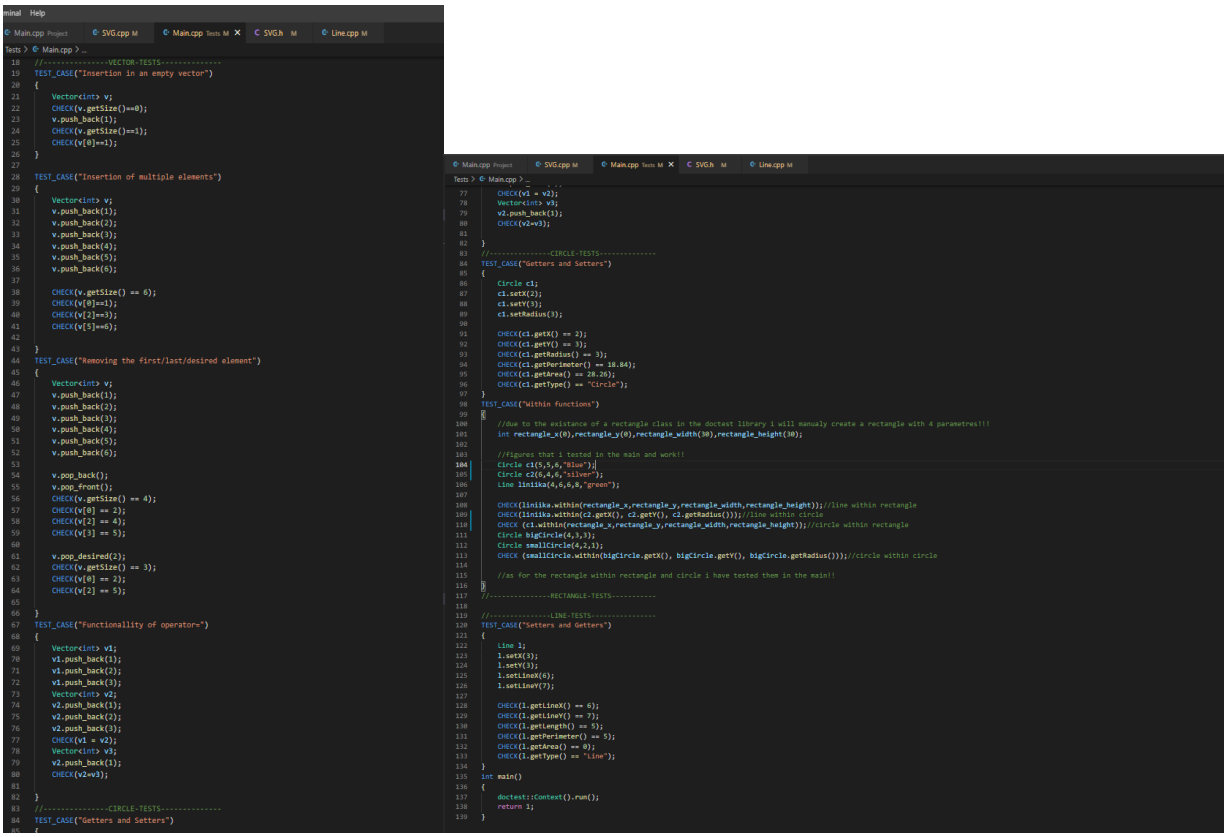
Той съдържа вектор от фигури, вектор от команди и името на файла.

Функциите, съдържащи се в SVG са насочени за изпълняване на основните команди на програмата.

Не бива да се забравя и споменаването на класовете Vector и String, които работят с динамична памет и спестяват изключително много работа и време.

4.2. *Тестове на проекта*

Основното тестване на методи/функции и т.н извърших в Main-а на програмата, но за отчетливост и красота на кода използвах библиотеката doctest, където написах тестове за съответните методи.



5.1. *Обобщение на изпълнението на началните цели*

Предвид фактът че Scannable Vector Graphics не са типично Cpp-ориентирани проектът адекватно приема/създава/запазва/изтрива/отпечатва и т.н фигури от даден файл. Началните цели, поставени ни по условие са изпълнени и командите на програмата успешно се справят с изискванията.

5.2. *Насоки за бъдещо развитие и усъвършенстване*

Проектът може да се разшири изключително много в много и разнообразни насоки. Самата възможност за добавяне на триизмерни фигури повишава сложността, потенциала и възможностите на програмата експонениално.

Добавянето на нови команди и методи като методи за “рисуване” на фигурите от файл със сигурност биха били зашеметяващо подобрение към интерфейса и т.н.