**Документация по проект №8 SVG файлове**

[Линк към GitHub репозиторито](https://github.com/KrashM/SVG_Editor)

**Глава 1: Увод**

***1.1*** ***Описание и идея на проекта***:

Програма, която да позволява лесен преглед и обработка на векторни изображения(SVG files). Потребителят може да влезе в приложението и както да преглежда вече създадени векторни изображения, така и да може да ги преправя и обработва по свой стил.

***1.2*** ***Цел и задачи на разработката***:

Целта е да се напише работещо приложение с лесен за работа потребителски интерфейс. В приложението да има едно основно меню. Където потребителят може да преглежда или обработва своите векторни изображения.

Задачи:

* Да се създадат основни класове
* Да се реализират необходимите структури от данни и алгоритми
* Да се направят валидации на данните
* Да се направи структура за данните
* Да се записва цялата информация от програмата във файлове
* Да се зарежда вече запазената информация от файловете
* Да се направи потребителски интерфейс

***1.3 Структура на документацията***:

Документацията е структурирана по следният начин:

* Увод
* Преглед на предметната област
* Проектиране
* Реализация, тестване
* Заключение

**Глава 2: Преглед на предметната област**

***2.1.* *Основни дефиниции, концепции и алгоритми, които ще бъдат използвани***

В проектът ще бъдат използвани работа с векторни (svg) файлове. Обработване на данни и записването им във файлове и след това четене от там. Също ще бъдат използвани вектори за съхранение на множество от еднотипна информация. Например всички фигури от файла. Ще се използва и String класа за по-лесно съхранение на последователност от символи. По естество ще има файлово разделение за по-добра структура и по-добро четене на кода, както и множество от функции.

***2.2. Дефиниране на проблеми и сложност на поставената задача***

Проблемите, за които програмата е създадена да решава са:

* Преглед на векторни изображения
* Обработване на файлове
* Добавяне на нови фигури

Сложността на поставената задача беше подходяща за дадения срок за нейното изпълнение.

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

За решаване на гореспоменатите проблеми, програмата използва множество функции и класове.

* Абстрактен клас Shape с поле за цвета на фигурата и множество от нейните точки.
  + Pure virtual функции за свойствата на фигурата (периметър, лице, лежи ли във фигура и т.н.)
  + Функция за транслация на фигурата
  + Функция за записване във файл
  + Функция за чете от файл
* Класът ShapeCollection има колекция от всички фигури
  + Функции за добавяне на фигура
  + Функция за транслиране на специфична фигура или на всички
  + Функция за изчистване на списъка
* Класът Vector е стандартна имплементация на динамичен масив от произволен тип(type T)
  + Функция за автоматично преоразмеряване
  + Функции за добавяне и търкане на елемент
  + Функции за търсене на елемент
* Класът String е стандартна имплементация на динамичен масив от символи (символен низ)
  + Функция за добавяне на елемент/и
  + Функции за търсене в низа
  + Функции за взимане на подниз
  + Предефинирани оператори за четене и писане на низа

**Глава 3: Проектиране**

***3.1. Обща архитектура***

Основни класове които се използват в последствие за структурата на други класове:

* Vector
* String
* Shape

Следващите класове са подредени по начина в който се използват един в друг:

* Shape – Има полета от тип String
* ShapeCollection – Колекция от тип Shape
* IO – Има статично поле от тип ShapeCollection

И накрая остава основния файл на програмата, а именно IO.cpp, която контролира входа и изхода(интерфейса на програмата) и също така останалите функционалности на платформата.

***3.2. Диаграми и слоеве с най-важните извадки от кода***

* Създаване на фигура:

void *IO*::create(){

*String* shape;

    cin >> shape;

    if(shape == "rectangle"){

*size\_t* x, y, width, height;

*String* color;

        cin >> x >> y >> width >> height >> color;

        if(!*IO*::colorNames.contains(color)){

            cout << "Invalid color name\n";

            return;

        }

*IO*::collection.addShape(**new** *Rectangle*(x, y, width, height, color));

    }

    else if(shape == "circle"){

*size\_t* x, y, radius;

*String* color;

        cin >> x >> y >> radius >> color;

        if(!*IO*::colorNames.contains(color)){

            cout << "Invalid color name\n";

            return;

        }

*IO*::collection.addShape(**new** *Circle*(x, y, radius, color));

    }

    else if(shape == "line"){

*size\_t* x1, y1, x2, y2;

*String* color;

        cin >> x1 >> y1 >> x2 >> y2 >> color;

        if(!*IO*::colorNames.contains(color)){

            cout << "Invalid color name\n";

            return;

        }

*IO*::collection.addShape(**new** *Line*(x1, y1, x2, y2, color));

    }

    cout << "Successfully created " << shape << " (" << *IO*::collection.size() << ")\n";

*IO*::changes = true;

}

* Четене на фигурите от файла:

*String* type;

*ifstream* ifs(*IO*::currentFile.c\_str());

const *size\_t* bufferSize = 1024;

char temp[bufferSize];

ifs.getline(temp, bufferSize);

ifs.getline(temp, bufferSize);

ifs.getline(temp, bufferSize);

while(true){

while(ifs.get() != '<');

        ifs >> type;

        if(type == "rect"){

*Rectangle* \*rect = **new** *Rectangle*();

            rect -> deserialize(ifs);

*IO*::collection.addShape(rect);

**delete** rect;

        }

        else if(type == "circle"){

*Circle* \*circle = **new** *Circle*();

            circle -> deserialize(ifs);

*IO*::collection.addShape(circle);

**delete** circle;

        }

        else if(type == "line"){

*Line* \*line = **new** *Line*();

            line -> deserialize(ifs);

*IO*::collection.addShape(line);

**delete** line;

        }

        else if(type == "/svg>") break;

    }

* Запазване на svg файла

void *IO*::save(){

*ofstream* ofs(*IO*::currentFile.c\_str(), *ios*::trunc);

    ofs << "<?xml version=\"1.0\" standalone=\"no\"?>\n"

        << "<!DOCTYPE svg PUBLIC \"-//W3C//DTD SVG 1.1//EN\"\"http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd\">\n"

        << "<svg>\n";

    for(*size\_t* i = 0; i < *IO*::collection.size(); i++)

*IO*::collection[i] -> serialize(ofs);

    ofs << "</svg>";

*IO*::saved = true;

*IO*::changes = false;

    ofs.close();

    cout << "Successfully saved the changes to " << *IO*::currentFile << '\n';

}

**Глава 4: Реализация, тестване**

***Управление на паметта и алгоритми. Оптимизации.***

Програмата е сведена до минимално опростен и лесен за написване начин. Всеки клас е дефиниран със своите полета и функционалности. Като за всеки реализиран метод са изпълнени стандарти, семантики и алгоритмите им са с оптимални сложности. Всеки клас отговаря собственоръчно за управлението на паметта си. Всеки един клас който използва динамична памет в последствие я освобождава при приключването на програмата.

**Глава 5. Заключение**

***5.1. Обобщение на изпълнението на началните цели***

Програмата се справя с работа с файлове. Успешно запазване на информацията във векторно изображение. Обработка на данните по векторното изображение. Принтиране на данните.

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

В бъдещи версии на програмата предстои да се оптимизират структурите използвани в програмата и алгоритмите, които те използват. Така че програмата да стане като прототип, не пълна версия, на известни други програми, създадени от известни компании.