# OOP Homework 2

## 引入

相信大家学到继承的时候都有看过类似“ `Circle` 和 `Rectangle` 继承自 `Shape` ”的例子，大家有没有想过把如果自己实现一个简单的图形绘制库，面向对象编程的编程范式将会是非常合适的选择。

这次的作业即是让大家实现一个简单的图形绘制库。因为不想让同学们纠结于怎么画图形（比如怎么画椭圆，怎么画贝塞尔曲线，怎么抗锯齿，怎么输出位图，……），所以大家的任务即是输出 SVG 格式的图片。

## SVG

SVG 格式是现在非常常用的矢量图片格式，而且在任何现代的浏览器上以及操作系统的预览功能中都可以正常地显示。一张 SVG 图片其实就是一个 XML 文件。例如

```
<svg version="1.1"
     baseProfile="full"
     width="300" height="200"
     xmlns="http://www.w3.org/2000/svg">
  <rect width="100%" height="100%" fill="red" />
  <circle cx="150" cy="100" r="80" fill="green" />
  <text x="150" y="125" font-size="60" fill="white">SVG</text>
</svg>
```

这样的一段代码就会产生如下的图片：


下面简单介绍一下 XML 的语法。

我们称 `<tag>` 这样的东西为标签，像上面代码就有 `<svg>` , `<rect>` , `<circle>` , `<text>` 这几

个标签。每一个标签都需要有一个结束标签。结束标签有两种形式。有的标签的内部是空的，那么结束标签就是在这个标签的右尖括号之前加上斜杠 `<tag />`，如上面代码中的 `<rect />`，`<circle />`，`<text />`。有的标签内部还有元素，比如上面的 `<svg>` 标签就包含了其他三个标签，对于这样的标签，它对应的结束标签就是 `</tag>`。

每一个标签都可以有自己的属性，跟在标签名的后面，键和值之间用一个等号相连，值必须用双引号包起来，每个键值对用空格分隔，如：`<tag key1="value1" key2="value2" />`

到这里为止，你应该能够读懂上面这段代码的结构了。

简单地理解，SVG 实际上就是在一个特定的规范下，制定了一些固定的标签以及他们的属性，而浏览器有符合规范的实现。这就是上面这段代码能够神奇的在浏览器中被现实为一张图片的原因。

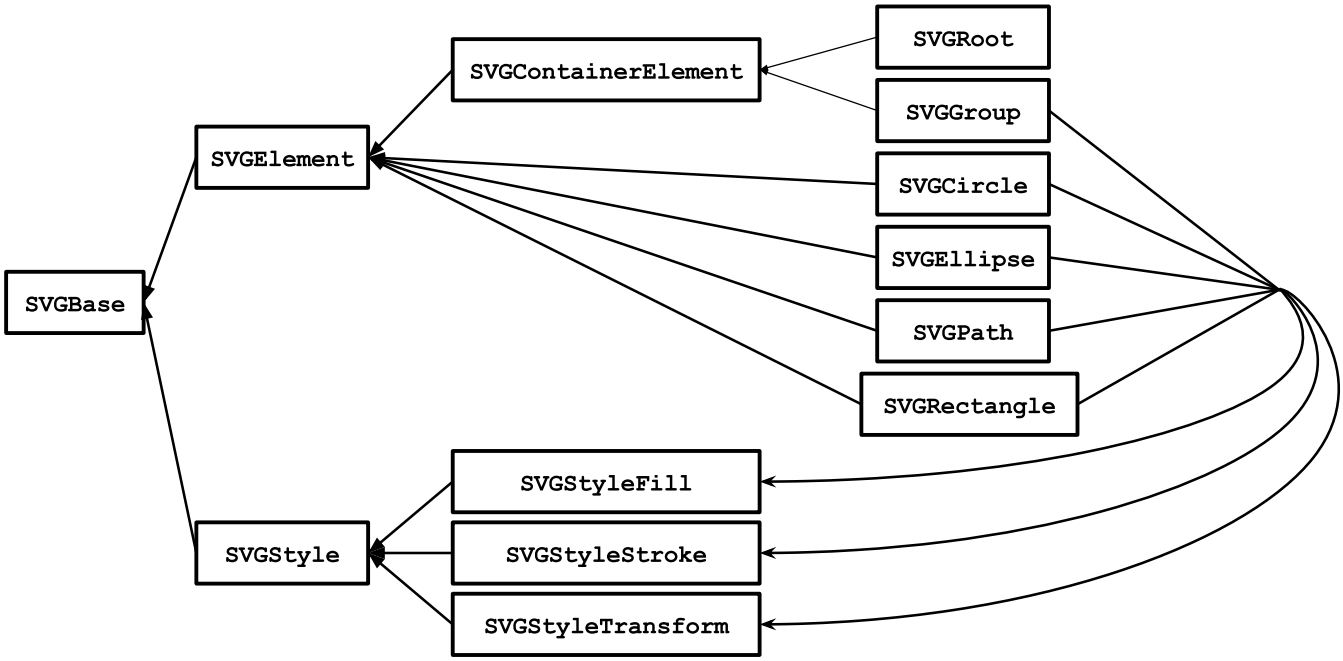
虽然这次作业是让大家实现图形绘制库，但是大家并不需要了解 SVG 如何使用，只需要有以上基本的 XML 知识即可。如果同学们对 SVG 感兴趣，可以在 https://developer.mozilla.org/en-US/docs/Web/SVG/Tutorial 看到一个非常简单易懂的介绍。

# 任务说明

大家需要完成下面的类的编写：

| 类名 | 说明 |
| --- | --- |
| SVGBase | SVG库的基类 |
| SVGElement | SVG元素的基类 |
| SVGContainerElement | SVG容器元素的基类 |
| SVGStyle | SVG样式的基类 |
| SVGCircle | 圆 `<circle>` |
| SVGEllipse | 椭圆 `<ellipse>` |
| SVGPath | 曲线路径 `<path>` |
| SVGRectangle | 矩形 `<rect>` |
| SVGGroup | 组合元素（容器） `<g>` |
| SVGRoot | SVG根元素（容器） `<svg>` |
| SVGStyleFill | 填充样式 |
| SVGStyleStroke | 描边样式 |
| SVGStyleTransform | 变换样式 |

类的继承关系如下图所示：




对于以上6种SVG元素的对象，它可以有一个全局唯一的 `id`，或者 `id` 为空。此外，每种元素会有一些不同的属性（属性含义略去不表，下面只写出它在 XML 代码中属性的名字）：

| 元素 | 属性 |
| --- | --- |
| `<circle>` | cx, cy, r |
| `<ellipse>` | cx, cy, rx, ry |
| `<path>` | d |
| `<rect>` | x, y, width, height, rx, ry |
| `<g>` | |
| `<svg>` | width, height |

以上属性的值除了 `<path>` 的 `d` 为字符串以外，其他的都可以当做是浮点数。

另外对于容器元素 `<svg>` 和 `<g>` 它们可以递归地包含以上6种元素。

对于这6种元素，每种元素都具有3类不同的样式，下面仅列出每一类样式的属性在 XML 代码中的名字：

| 样式 | 属性 |
| --- | --- |
| 填充 | fill, fill-opacity |
| 描边 | stroke, stroke-opacity, stroke-width |
| 变换 | transform |

以上属性的值除了 `fill-opacity`，`stroke-opacity`，`stroke-width` 是浮点数以外，其他的都可以当做是字符串。

# 任务要求

- 能用 `SVGRectangle rect("my-rectangle");` 建立一个带id属性的元素
- 能用 `SVGRectangle rect;` 建立一个id为空的元素
- 能用 `rect.setSize(200.5, 300.5);` 设置元素的特有属性
- 能用 `rect.setStrokeColor("#000");` 设置元素风格
- 能用 `container.add(rect);` 将元素添加到容器元素中
- 能用 `out << container` 将元素输出到流中
- 能用 `container.findContainer("id");` 返回一个容器元素中具有特定id的容器元素的引用
- 能用 `container.remove("id");` 删除一个容器元素中具有特定id的元素（不需要支持删除子容器中元素）
- 能用 `rect.clone()` 返回一个元素的深拷贝（即如果这个元素是一个容器，那么除了这个元素本身，需要递归地拷贝它包含的元素），返回值具有 `SVGElement*` 类型
- 能用 `container.clone_as_container()` 返回一个容器元素的深拷贝，返回值具有 `SVGContainerElement*` 类型
- 以上要求中的 `SVGRectangle` 可以指代任何元素； `rect` 可以指代任何元素的对象； `container` 可以指代任何容器元素的对象； `out` 可以指代任何输出流； `set***` 可以指代任何特有属性的设置
- 对于4个基类，将其实现为抽象类，即不允许构建这4个抽象类的对象
- 能够编译运行 `src/` 文件夹中的程序，并且产生出与 `examples/` 文件夹中对应图片相同的图片（目测即可），**注意每个元素只能被输出恰好一次**（比如一个错误的程序可能对同一个 `<g>` 输出了两次）
- **没有内存泄漏**

# Don't Panic!

这真的是一个小作业，不是大作业。

# 建议编写顺序

1. 编写 `SVGBase` 和 `SVGElement` 类，之后编写派生类 `SVGRectangle`，测试 `<rect>` 元素的输出是否正常。
2. 编写 `SVGStyle` 和 `SVGStyleFill` 类，将其加入 `SVGRectangle` 类的基类列表中，测试 `<rect>` 元素的样式设置和输出是否正常。
3. 类似 `SVGStyleFill`，完成 `SVGStyleStroke` 和 `SVGStyleTransform` 类，并加入 `SVGRectangle` 类的基类列表中，测试 `<rect>` 元素的相关输出是否正常。注意到此时 `SVGRectangle` 元素已经完成。
4. 类似 `SVGRectangle`，完成 `SVGCircle`，`SVGEllipse`，`SVGPath` 类。
5. 编写 `SVGContainerElement` 类，之后编写派生类 `SVGRoot`，尝试将其他元素加入 `<svg>` 并检验输出。注意到这个时候你已经能输出一个完整的 SVG 图片了。
6. 类似 `SVGRoot`，完成 `SVGGroup` 类，测试元素的递归添加是否有问题。

# 小提示

- 每个类的头文件已经为大家建立好了，并且其中包含了大量的代码片段，实际上你只需要在有注释的地方**填空**即可。
- 想想 `clone()` 会发挥什么重要作用？对于一个容器元素，它和 `clone_as_container()` 有什么区别？ `clone_body_from()` 有什么用？
- `<path>` 的 `d` 属性可以使用一个 `std::ostringstream` 保存，要输出时使用 `str()` 成员函数转换为字符串即可
- 容器元素中可以使用 `std::list` 保存容器中的元素的基类指针
- 容器元素中可以使用 `std::map` 保存id对应元素的 `std::list::iterator`
- `delete` 除了在析构函数中出现以外，还会在 `.remove()` 函数中出现
- 代码量**可能**较大，请勿拖延
- 代码量可能较大，但是按照建议编写顺序编写的话，应该是比较容易的
- 如果遇到了什么不懂的问题，请及时和同学以及助教讨论，请勿拖延 (再一次强调)
- 你可以在 http://cplusplus.com 查看 C++ Reference，例如：http://cplusplus.com/map

# 考察点

- 继承、多重继承
- 构造函数、析构函数
- 虚函数
- 查看 Reference 的能力
- 阅读代码能力
- ~~长文阅读能力 与 心理素质 与 是否有拖延症~~

# 提交方式

- 将 `lib/` 文件夹打包
- 发送至 `i+oop2@abcdabcd987.com`