CIS*2750
Assignment 2
Deadline: Saturday, February 15, 9am
Weight: 17%

Assignment 2 will consist of three modules. To simplify regression testing of our Assignment 1 code, we will not be updating any Assignment 1 functionality.

Assignment 2 modules:

1. Functions for creating valid SVGimages, validating SVGimages, and writing SVGimages to a file.
2. Functions for adding components to SVGimages.
3. A set of "glue" functions to convert an SVGimage and its components into JSON strings - and vice versa. Some of these functional will be optional, and form a bonus mark. However, all of these functions will be useful in later assignments: they will help us integrate the C library created in in A1/A2 with the server-side JavaScript code in A3/A4.

You are provided with a temporary header file for A2 Module 1 (`SVGParser_A2temp.h`). This header will be updated when Modules 2 and 3 come out. Once Module 3 is released, I will post the final official header file for Assignment 2, which will be used in the A2 test harness for grading.

Keep in mind that functions in Modules 2 and 3 are shorter than those in Module 1, and Modules 2 and 3 may be released at the same time.

Module 1 functionality

```
SVGimage* createValidSVGimage(char* fileName, char* schemaFile);
```

An updated version of `createSVGimage`. The only difference is that this function validates the `xmlDoc` returned by the libxml2 function `xmlReadFile` against an XSD file that represents the SVG standard before doing any further parsing and building the `SVGimage`.

This function does the parsing and allocates a SVG image object. It accepts the name of the SVG file, and the name of the XSD file. If the file contains a valid SVG image and has been parsed successfully, a pointer to the to the newly created `SVGimage` object is returned. If the parsing fails for any reason - invalid XML format, invalid SVG format, etc. - the function must return `NULL`.

```
bool writeSVGimage(SVGimage* doc, char* fileName);
```

This function takes an `SVGimage` struct and saves it to a file in SVG format. Its arguments are an `SVGimage` and the name of a new file. It must return `true` if the write was successful, and `false` is the write failed for any reason - invalid output file name, etc.. This function can assume that the `doc` argument has already been validated with `validateSVGimage`, which is discussed below. It must still make sure the arguments are not NULL.

```
bool validateSVGimage(SVGimage* doc, char* schemaFile);
```

This function takes an `SVGimage` and the name of a valid SVG schema file, and validates the contents of the `SVGimage` against an XSD file that represents the SVG standard. It also validated the contents against the constraints specified in `SVGParser.h`. It returns `true` if the `SVGimage` contains valid data, and `false` otherwise.

There are two aspects to `SVGimage` validity. First, its contents must represent a valid SVG image once converted to XML. This can be validated using a method similar to what do in `createValidSVGimage`.

The second aspect is whether the `SVGimage` violates any of the constraints specified in the `SVGparser.h`. Some of these constraints reflect the SVG specification. For example, the SVG documentation states that a circle radius cannot be negative. However, validating a libxml tree against the SVG schema file will not catch this violation - as long as the radius is a valid number, libxml will consider the underlying XML document to be valid and fully compliant with the schema.

In addition, there are constraints that enforce the internal consistency of the data structures in the `SVGimage` - for example, all pointers in an `SVGimage` must be initialized and must not be NULL.

This means that `validateSVGimage` must manually check the constraints of the struct against the specifications listed in `SVGParser.h` - ensure that the numbers are within valid ranges, lists are not NULL, etc.

Creating an XML tree

This might seem like a lot of work, but the libxml library has a number of functions that can help you. Most importantly, it has functions for:
- Writing a libxml tree to an XML file
- Validating a libxml tree against a schema file

In both cases, the libxml tree is represented as an `xmlDoc` struct - just like the one you get from `xmlReadFile` in Assignment 1. As a result, functions `writeSVGimage`, `createValidSVGimage`, and `validateSVGimage` become quite short and simple once you create two helper functions:
- A function that converts an `SVGimage` into an `xmlDoc` struct, i.e. a libxml tree. This tree can be then be easily saved to disk or validated against an XSD file using libxml functionality.
- A function that validates a libxml tree.

Keep in mind that the order of elements in an SVG image matters. For example, if a circle is in the SVG file one line after a rectangle, the circle is drawn on top of the rectangle. In A1, when we parsed the SVG contents into separate lists, we have lost some of this order.

Solving this problem fully would require a different set of data structures, so we will make a simplifying assumption. In all SVG files that we read/write, the order will be as follows:
- Children of `<svg>` node:
  1. `<rect>`
  2. `<circle>`
  3. `<path>`
  4. `<g>`
- Children of `<g>` node:
  1. `<rect>`
  2. `<circle>`
  3. `<path>`
  4. `<g>`

In other words, in the input file, all rectangles will always come before circles, circles - before paths, etc..

As a result, when creating an XML tree, you must always write the `SVGimage` elements in the same order. For example, if the original SVG image contains multiple paths followed by multiple groups (e.g. `Emoji_poo.svg`), you would add contents of `SVGimage->paths` to the XML tree first, followed by the contents of `SVGimage->groups`. Same applied to groups: when creating a group element in the XML tree, we would first add all child rectangles to it, then child circles, etc..

As with Assignment 1, the libxml2 documentation has some useful examples that can get you started:
- Creating an XML tree struct - i.e. an XML tree: http://www.xmlsoft.org/examples/tree2.c
- Saving an XML tree to a file: http://www.xmlsoft.org/examples/tree2.c

- Validating an XML tree against a schema file: http://knol2share.blogspot.com/2009/05/validate-xml-against-xsd-in-c.html

Useful documentation (in addition to parser.h on the libxml2 site):
- tree.h contains the functions for navigating and creating XML tree elements: http://www.xmlsoft.org/html/libxml-tree.html
- xmlSchemaTypes.h contains functions for working with XML schema: http://www.xmlsoft.org/html/libxml-xmlschemastypes.html

NOTE: to set the namespace in an xml tree, use two functions:
- first use `xmlNewNs` to create the namespace (set the prefix to NULL)
- then use `xmlSetNs` to set the namespace for the root node