CIS*2750
Assignment 2
Module 3

The functions in this module will provide the interface between our parser API, which is written in C, and the web-based GUI in A3 and A4, which will rely on JavaScript and HTML. The different components will communicate using strings in JavaScript Object Notation (JSON) format. We will discuss the JSON format in more detail in a later class. For now, the output format will be provided for you.

These functions will allow your A3 to work with existing SVGimage objects from a Web GUI, and save new / updated objects to disk. Most of these functions are designed to convert the SVGimage and its components into JSON strings - and vice versa. You will expand on these functions - and compile them with others A1/A2 functions - in A3.

The function descriptions in this document are quite long (and repetitive), for the sake of clarity and precision. However, most the functions themselves will be fairly short and simple, and many will closely resemble the various `print...` functions you have implemented in Assignment 1. They are also quite similar to each other.

Please pay careful attention to quotes, spaces, and other details in the output functions. They are important.

New functions

```
char* attrToJSON(const Attribute *a);
char* circleToJSON(const Circle *c);
char* rectToJSON(const Rectangle *r);
char* pathToJSON(const Path *p);
char* groupToJSON(const Group *g);

char* attrListToJSON(const List *list);
char* circListToJSON(const List *list);
char* rectListToJSON(const List *list);
char* pathListToJSON(const List *list);
char* groupListToJSON(const List *list);

char* SVGtoJSON(const SVGimage* imge);
```

Bonus functions - will be released in a separate module, not required for A2, but will be very useful for A3)

```
SVGimage* JSONtoGPX(const char* svgString);
Rect* JSONtoRect(const char* svgString);
Circle* JSONtoCircle(const char* svgString);
```

Part 1 - StructToJSON functions

1. *attrToJSON*

```
char* attrToJSON(const Attribute *a);
```

Converts an `Attribute` struct to a string in JSON format. The function must return a newly allocated string in the following format:

```
{"name":"attrName","value":"attrVal"}
```

where `attrName` is the name of the attribute, and `attrVal` is its value. Note the quotes in the string - they are part of the format.

For example, given an Attribute with the name `fill`, and value `red` the resulting string would be:
`{"name":"fill","value":"red"}`

The format **must** be exactly as specified. Do not add any spaces, newlines, or change capitalization.

The returned string contents for this function - and all the other `...ToJSON` functions below - will contain double-quote characters, so you will need to use the escape sequence `\"` in your code.

This function must not modify its argument in any way.

If the argument `a` is NULL, the function must return the string `{}` (there is no space there - just two chars).

2. *circleToJSON*

```
char* circleToJSON(const Circle *c);
```

Converts a `Circle` struct to a string in JSON format. The function must return a newly allocated string in the following format (note the presence/absence of quotes around values):

```
{"cx":xVal,"cy":yVal,"r":rVal,"numAttr":attVal,"units":"unitStr"}
```

where:
- `xVal` is the centre x coordinate
- `yVal` is the centre y coordinate
- `rVal` is radius
- `attVal` is the number of elements in the otherAttributes list of that circle
- `unitStr` is the units

For example:
- given a `Circle` created from `<circle cx="32" cy="32" r="30" fill="#ffdd67"/>`, the corresponding string would be
  `{"cx":32,"cy":32,"r":30,"numAttr":1,"units":""}`
- given a `Circle` created from `<circle cx="32cm" cy="32cm" r="30cm"/>`, the corresponding string would be
  `{"cx":32,"cy":32,"r":30,"numAttr":0,"units":"cm"}`

This function must not modify its argument in any way.

If the argument `c` is NULL, the function must return the string `{}` (there is no space there - just two chars).

3. *rectToJSON*

```
char* rectToJSON(const Rectangle *r);
```

Converts a `Rectangle` struct to a string in JSON format. The function must return a newly allocated string in the following format:

```
{"x":xVal,"y":yVal,"w":wVal,"h":hVal,"numAttr":attVal,"units":"unitStr"}
```

where:
- `xVal` is the x coordinate
- `yVal` is the y coordinate
- `wVal` is the width
- `hVal` is the height

- `attVal` is the number of elements in the otherAttributes list of that rectangle
- `unitStr` is the units

For example:
- given a `Rectangle` created from
  `<rect x="1cm" y="1cm" width="19cm" height="15cm" fill="none" stroke="blue" stroke-width="1" />` the corresponding string would be
  `{"x":1,"y":2,"w":19,"h":15,"numAttr":3,"units":"cm"}`
- given a `Rectangle` created from
  `<rect width="2" height="2"/>` the corresponding string would be
  `{"x":0,"y":0,"w":2,"h":2,"numAttr":0,"units":""}`

This function must not modify its argument in any way.

If the argument `r` is NULL, the function must return the string `{}` (there is no space there - just two chars).

4. pathToJSON

```
char* pathToJSON(const Path *p);
```

Converts a `Path` struct to a string in JSON format. The function must return a newly allocated string in the following format:

```
{"d":"dVal","numAttr":attVal}
```

where:
- `dVal` is the path data. `dVal` must be no more than 64 characters long. If the actual path data is longer, truncate the path after the first 64 characters. he quotes before/after `dVal` do not count towards the 64-character limit.
- `attVal` is the number of elements in the otherAttributes list of that path

For example:
- given a `Path` created from
  `<path d="m47 36c-15 0-15 0-29.9 0-2.1 0-2.1 4-.1 4 10.4 0 19.6 0 30 0 2 0 2-4 0-4" fill="#fff"/>` the corresponding string would be
  `{"d":"m47 36c-15 0-15 0-29.9 0-2.1 0-2.1 4-.1 4 10.4 0 19.6 0 30 0 2 0","numAttr":1}`
  Note that the path is truncated to be 64 characters long
- given a `Path` created from
  `<path d="m47 36c-15 0-15 0-29.9 0-2.1 0-2.1 4-.1 4/>` the corresponding string would be
  `{"d":"m47 36c-15 0-15 0-29.9 0-2.1 0-2.1 4-.1 4","numAttr":0}`

This function must not modify its argument in any way.

If the argument `p` is NULL, the function must return the string `{}` (there is no space there - just two chars).

5. groupToJSON

```
char* groupToJSON(const Group *g);
```

Converts a `Group` struct to a string in JSON format. The function must return a newly allocated string in the following format:

```
{"children":cVal,"numAttr":attVal}
```

where:
- `cVal` is the total number of immediate children of the group - i.e. the sum of the lengths of the 4 list containing the children.
- `attVal` is the number of elements in the otherAttributes list of that path

For example:
- given the first group in an `SVGimage` created from `quad01.svg`, which contains a rectangle and a path, and has a and a `fill` attribute, the corresponding string would be
  `{"children":2,"numAttr":1}`

This function must not modify its argument in any way.

If the argument `g` is NULL, the function must return the string `{}` (there is no space there - just two chars).

6. *SVGtoJSON*

```
char* SVGtoJSON(const SVGimage* image);
```

Converts a `SVGimage` struct to a string in JSON format. The function must return a newly allocated string in the following format:

```
{"numRect":numR,"numCirc":numC,"numPaths":numP,"numGroups":numG}
```

where
- `numR` is the total number of Rectangles in the `SVGimage`
- `numC` is the total number of Circles in the `SVGimage`
- `numP` is the total number of Paths in the `SVGimage`
- `numG` is the total number of Groups in the `SVGimage`

For example, given the `SVGimage` created from `quad01.svg`, we have
- 1 Rectangle
- 5 circles
- 2 paths
- 3 groups

The output string would be:
`{"numRect":1,"numCirc":5,"numPaths":2,"numGroups":3}`

The format must be exactly as specified. Do not add any spaces, newlines, or change capitalization. As always, pay close attention to the quotes.

This function must not modify its argument in any way.

If the argument `image` is NULL, the function must return the string `{}` (there is no space there - just two chars).

NOTE: the various `get...` functions from A1 Module 2 are very useful here. You just have to remember to clear the lists correctly, so the `SVGtoJSON` function does not leak memory. The easiest way to do so is, when creating the `List` in the appropriate `get...` function, to pass a stub for the corresponding `delete...` function. This way, you can, for example, free the list of all circles without deleting the Circle structs that the list points to - you just have to pass a stub that frees noting instead of `deleteCircle` when initializing the list inside `getCircles`.

Part 2 - ListToJSON functions

7. *attrListToJSON*

```
char* attrListToJSON(const List *list);
```

This function will convert a list of Attributes - e.g. the `otherAttributes` list of a `SVGimage`, or `otherAttributes` list of a `Circle` - into a JSON string. You can - and should - use `attrToJSON` function defined above.

The function `attrListToJSON` must return a newly allocated string in the following format:

```
[AttrString1,AttrString2,…,AttrStringN]
```

where every `AttrString` is the JSON string returned by `attrToJSON`, and `N` is the number of attributes in the original list. The order of `AttrString`s must be the same as the order of attributes in the original list.

For example, given the `otherAttributes` list from a `Rectangle` created from
`<rect x="1cm" y="1cm" width="19cm" height="15cm" fill="none" stroke="blue" stroke-width="1" />`

The resulting string would be:
`[{"name":"fill","value":"none"},{"name":"stroke","value":"blue"},{"name":"stroke-width","value":"1"}]`

Please note that the string above has no newlines; it is spread over multiple lines for readability. The actual string will contain no newlines or spaces, and look like this (sorry for the teeny font):

```
[{"name":"fill","value":"none"},{"name":"stroke","value":"blue"},{"name":"stroke-width","value":"1"}]
```

The format **must** be exactly as specified. Do not add any spaces or newlines.

Do not modify the order of elements in the original list. Also, do not make any assumptions about the length of the list - it can contain any number of elements.

This function must not modify its argument in any way.

If the argument `list` is NULL, or an empty list, the function must return the string `[]` (there is no space there - just two chars).

8. *circListToJSON*

```
char* circListToJSON(const List *list);
```

This function will convert a list of Circles into a JSON string. You can - and should - use `circToJSON` function defined above.

The function `circListToJSON` must return a newly allocated string in the following format:

```
[CircString1,CircString2,…,CircStringN]
```

where every `CircString` is the JSON string returned by `circToJSON`, and `N` is the number of circles in the original list. The order of `CircString`s must be the same as the order of attributes in the original list.

For example, given a with two circles:
- a `Circle` created from `<circle cx="32" cy="32" r="30" fill="#ffdd67"/>`
- a `Circle` created from `<circle cx="32cm" cy="32cm" r="30cm"/>`

The corresponding string would be
`[ { " c x " : 3 2 , " c y " : 3 2 , " r " : 3 0 , " n u m A t t r " : 1 , " u n i t s " : " " } , {"cx":32,"cy":32,"r":30,"numAttr":0,"units":"cm"}]`

As before, the string above has no newlines; it is spread over multiple lines for readability. The actual string will contain no newlines or spaces, and look like this (sorry for the teeny font):

```
[{"cx":32,"cy":32,"r":30,"numAttr":1,"units":""},{"cx":32,"cy":32,"r":30,"numAttr":0,"units":"cm"}]
```

The format **must** be exactly as specified. Do not add any spaces or newlines.

Do not modify the order of elements in the original list. Also, do not make any assumptions about the length of the list - it can contain any number of elements.

This function must not modify its argument in any way.

If the argument `list` is NULL, or an empty list, the function must return the string `[]` (there is no space there - just two chars).

9. *rectListToJSON*

```
char* rectListToJSON(const List *list);
```

This function will convert a list of Rectangles into a JSON string. You can - and should - use `rectToJSON` function defined above.

The function `rectListToJSON` must return a newly allocated string in the following format:

```
[RectString1,RectString2,…,RectStringN]
```

where every `RectString` is the JSON string returned by `rectToJSON`, and `N` is the number of rectangles in the original list. The order of `RectString`s must be the same as the order of attributes in the original list.

For example, given the list with two rectangles:

1. a `Rectangle` created from
   `<rect x="1cm" y="1cm" width="19cm" height="15cm" fill="none" stroke="blue" stroke-width="1" />`
2. a `Rectangle` created from
   `<rect width="2" height="2"/>`

The corresponding string would be
`[ { " x " : 1 , " y " : 2 , " w " : 1 9 , " h " : 1 5 , " n u m A t t r " : 3 , " u n i t s " : " c m " } , {"x":0,"y":0,"w":2,"h":2,"numAttr":0,"units":""}]`

As before, the string above has no newlines; it is spread over multiple lines for readability. The actual string will contain no newlines or spaces, and look like this (sorry for the teeny font):

```
[{"x":1,"y":2,"w":19,"h":15,"numAttr":3,"units":"cm"},{"x":0,"y":0,"w":2,"h":2,"numAttr":0,"units":""}]
```

The format **must** be exactly as specified. Do not add any spaces or newlines.

Do not modify the order of elements in the original list. Also, do not make any assumptions about the length of the list - it can contain any number of elements.

This function must not modify its argument in any way.

If the argument `list` is NULL, or an empty list, the function must return the string `[]` (there is no space there - just two chars).

10. *pathListToJSON*

```
char* pathListToJSON(const List *list);
```

This function will convert a list of Paths into a JSON string. You can - and should - use `pathToJSON` function defined above.

The function `pathListToJSON` must return a newly allocated string in the following format:

```
[PathString1,PathString2,…,PathStringN]
```

where every `PathString` is the JSON string returned by `pathToJSON`, and `N` is the number of paths in the original list. The order of `PathString`s must be the same as the order of attributes in the original list.

For example, given the list with two paths:

- a `Path` created from
  `<path d="m47 36c-15 0-15 0-29.9 0-2.1 0-2.1 4-.1 4 10.4 0 19.6 0 30 0 2 0 2-4 0-4" fill="#fff"/>`
- a `Path` created from
  `<path d="m47 36c-15 0-15 0-29.9 0-2.1 0-2.1 4-.1 4/>`

The corresponding string would be

```
[{"d":"m47 36c-15 0-15 0-29.9 0-2.1 0-2.1 4-.1 4","numAttr":0},{"d":"m47 36c-15 0-15 0-29.9 0-2.1 0-2.1 4-.1 4 10.4 0 19.6 0 30 0 2 0","numAttr":1}]
```

As before, the string above has no newlines; it is spread over multiple lines for readability. The actual string will contain no newlines or spaces, and look like this (sorry for the teeny font):

```
[{"d":"m47 36c-15 0-15 0-29.9 0-2.1 0-2.1 4-.1 4","numAttr":0},{"d":"m47 36c-15 0-15 0-29.9 0-2.1 0-2.1 4-.1 4 10.4 0 19.6 0 30 0 2 0","numAttr":1}]
```

The format **must** be exactly as specified. Do not add any spaces or newlines.

Do not modify the order of elements in the original list. Also, do not make any assumptions about the length of the list - it can contain any number of elements.

This function must not modify its argument in any way.

If the argument `list` is NULL, or an empty list, the function must return the string `[]` (there is no space there - just two chars).

11. *groupListToJSON*

```
char* groupListToJSON(const List *list);
```

This function will convert a list of Groups into a JSON string. You can - and should - use `groupToJSON` function defined above.

The function `groupListToJSON` must return a newly allocated string in the following format:

```
[GroupString1,GroupString2,…,GroupStringN]
```

where every `GroupString` is the JSON string returned by `groupToJSON`, and `N` is the number of groups in the original list. The order of `GroupString`s must be the same as the order of attributes in the original list.

For example, given the group list in an `SVGimage` created from `quad01.svg` (see file for details) the corresponding string would be
`[{"children":2,"numAttr":1},{"children":3,"numAttr":1},{"children":2,"numAttr":1}]`

The format **must** be exactly as specified. Do not add any spaces or newlines.

Do not modify the order of elements in the original list. Also, do not make any assumptions about the length of the list - it can contain any number of elements.

This function must not modify its argument in any way.

If the argument `list` is NULL, or an empty list, the function must return the string `[]` (there is no space there - just two chars).