CIS*2750
Assignment 1

Module 2: accessor and search functions

The purpose of the parser we are building in A1 is to extract the major components from a vector image file, and fill in our data structures, which will be used in later assignments. Make sure that, in addition to the SVG specification, you also carefully read the comments in `SVGParser.h` - each data structure includes detailed comments indicating the constraints for various components.

The functions below will allow us to examine the internal state of the `SVG` struct and its components, and modify them in later assignments.

Required Functions - accessors

This set of four functions will act somewhat similar to accessors (get...) methods of a class. We will use them to return a list of all specific primitives. Keep in mind that we do not want to replicate the primitives. So if we have an image with two circles, we will return a list of pointers to the existing circles, rather than a list containing newly allocated copies of the circles. This way we can also modify the image contents - for example, we could scale all rectangles by iterating through the list returned by `getRects` and multiplying the width and height by 2.

```
List* getRects(SVG* img);
```

Function that returns a list of all rectangles in the `SVG` struct. If there are none, it returns an empty list (not a NULL pointer).

```
List* getCircles(SVG* img);
```

Function that returns a list of all circles in the `SVG` struct. If there are none, it returns an empty list (not a NULL pointer).

```
List* getGroups(SVG* img);
```

Function that returns a list of all groups in the `SVG` struct. If there are none, it returns an empty list (not a NULL pointer).

```
List* getPaths(SVG* img);
```

Function that returns a list of all paths in the `SVG` struct. If there are none, it returns an empty list (not a NULL pointer).

In all of the above functions, if `img` is NULL, return NULL.

Please keep in mind that you need to get a list of pointers to all structs of the specified kind in the `SVG` struct. - the ones from the lists in the `SVG` struct, as well as the ones from the Groups wishing that SVG struct For example, if you use `createSVG` to create an `SVG` struct from `quad01.svg` (once of the sample files posted for A1), then:

- `getRects` would return a list with 1 rectangle: 0 from `SVG->rectangles`, 1 from the first group

- `getCircles` would return a list with 5 circles: 0 from `SVG->circles`, 3 from the second group (the black ones), and 2 from the second group (the grey ones)
- `getGroups` would return a list with 3 groups, all from `SVG->groups`
- `getPaths` would return a list with 2 paths: 1 from `SVG->paths` and 1 from the first group

Required Functions - summaries

This set of five functions will tell us something about the internal structure of the image. We will use them to find if the image contains specific elements. Unfortunately elements in an SVG file are not uniquely named, so we have to search by their properties. To make your life a little simpler, the API requires you to return the count of elements with the search criterion (i.e. 0 or more), rather than a list containing a pointer to them.

This will be much easier to implement if you implement some sort of a generic search function, similar to the `findElement` function in your List API. In fact, you will be able to use `findElement` as a template to help you create your implementation.

All of the `num...` functions below must return 0 if their arguments are invalid - e.g. NULL pointers, empty strings, negative area/length, etc..

```
int numRectsWithArea(SVG* img, float area);
```

Function that returns the number of all rectangles with the specified area. Return 0 if no such rectangles are found, or if any of the arguments are invalid.

```
int numCirclesWithArea(SVG* img, float area);
```

Function that returns the number of all circles with the specified area. Return 0 if no such circles are found, or if any of the arguments are invalid.

```
int numPathsWithdata(SVG* img, char* data);
```

Function that returns the number of all paths with the specified data. Return 0 if no such paths are found, or if any of the arguments are invalid.

```
int numGroupsWithLen(SVG* img, int len);
```

Function that returns the number of all groups with the specified length. We will define group length as:

length of the Group->rectangles list
+
length of the Group->circles list
+
length of the Group->paths list
+
length of the Group->groups list

So if a group has 0 circles, 2 rectangles, 1 subgroup, and 5 paths, its length is 8. Please keep in mind that we don't care how many elements the subgroups contain - we simply count the number of subgroups. Return 0 if no such circles are found, or if any of the arguments are invalid.

```
int numAttr(SVG* img);
```

Function that returns the number of `Attributes` in all structs of the `SVG`. For example, the `SVG` created from `Emoji_poo.svg` should have the total of 9 `Attribute` structs in it. The total number of XML element attributes is larger than that, but many of these attributes get their own fields in their respective structs. For example, the values attribute `d` for `path` SVG elements becomes `Path.data` in our code, while all the other attributes of the original `path` element are stored as `Attribute` structs in the `Path.otherAttributes` list. Return 0 if no such circles are found, or if any of the arguments are invalid.

NOTE: keep in mind that some of the functions above will need to compute a float value, and then compare it to the argument - another float - for equality. We do not want to mess around with float equality tests. To simplify things, you will convert them to ints by applying the ceil() function (in math.h) to both values before comparison.

For example, let's say you need to find all rectangles of width 20. Your image contains 2 rectangles:
- One with width=3.5, height = 4.5
- One with width=4.3, height = 4.5

someone calls the function: numRectsWithArea(someImage, 20);

Rectangle 1 has area 3.5 * 4.5 = 15.75. ceil(15.75) = 16, and ceil(20) = 20, so rectangle 1 does not match.
Rectangle 2 has area 4.3 * 4.5 = 19.35. ceil(19.35) = 20, and ceil(20) = 20, so rectangle 2 is a match, and hasRectsWithArea returns 1.