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 GPXdoc objects from a Web GUI, and save new / updated objects to disk. Most of these functions are designed to convert the GPXdoc and its components into JSON strings - and vice versa. We also have a function for adding components to existing structs. You will expand on these functions in A3.

The function descriptions in this document are quite long, 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.

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

New functions

Required

```
char* routeListToJSON(const List *list);
char* trackListToJSON(const List *list);
char* trackToJSON(const Track *tr);
char* routeToJSON(const Route *rt);
char* GPXtoJSON(const GPXdoc* gpx);
```

Bonus

```
void addWaypoint(Route *rt, Waypoint *pt);
void addRoute(GPXdoc* doc, Route* rt);
GPXdoc* JSONtoGPX(const char* gpxString);
Waypoint* JSONtoWaypoint(const char* gpxString);
Route* JSONtoRoute(const char* gpxString);
```

Description

1. *routeToJSON*

```
char* routeToJSON(const Route *rt);
```

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

```
{"name":"routeName","numPoints":numVal,"len":routeLen,"loop":loopStat}
```

Notes:
- If the route has no name, the `routeName` should be `None`
- `loopStat` is either `true` or `false`
- Notice that `numVal`, `routeLen`, and `loopStat` do not have quotes around them
- numVal should be rounded to the nearest 10m, and be displayed with 1 value (0) after the decimal point

For example:
- given a Route with the name `Some route`, 3 points (so it cannot be a loop), and total length of 160m, the resulting string would be:
  `{"name":"Some route","numPoints":3,"len":160.0,"loop":false}`

- given a loop Route with no name, 5 points, and total length of 736.7m, the resulting string would be:
  `{"name":"None","numPoints":5,"len":740.0,"loop":true}`
  Note that the route length is rounded - use the function created in Module 2 here

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

2. *trackToJSON*

```
char* trackToJSON(const Track *tr);
```

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

```
{"name":"routeName","len":routeLen,"loop":loopStat}
```

This is very similar to the previous function, except we do not include the point count (or segment count, which might also be expected).

For example, a non-loop track called Mount Steele Trail with the length of 340m would look like this:
`{"name":"Mount Steele Trail","len":340.0,"loop":false}`

All the formatting details described above still apply. As with the previous function, this distance is rounded to the nearest 10m

This function must not modify its argument in any way.

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

3. *routeListToJSON*

```
char* routeListToJSON(const List *routeList);
```

This function will convert a list of Routes - e.g. the routes list of a `GPXdoc`, or a list returned by `getRoutesBetween`- into a JSON string. You can - and should - use `routeToJSON` function defined above.

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

```
[RouteString1,RouteString2,…,RouteStringN]
```

where every `RouteString` is the JSON string returned by `routeToJSON`, and `N` is the number of events in the original list. The order of `RouteString`s must be the same as the order of routes in the original list.

For example, assume that the list contains two Routes
- given a Route with the name `Some route`, 3 points (so it cannot be a loop), and total length of 160m
- given a loop Route with the name `Route 2`, 5 points, and total length of 736.7m

The resulting string would be:
`[{"name":"Some route","numPoints":3,"len":160.0,"loop":false},{"name":"Route 2","numPoints":5,"len":740.0,"loop":true}]`

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":"Some route","numPoints":3,"len":160.0,"loop":false},{"name":"Route 2","numPoints":5,"len":740.0,"loop":true}]
```

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

4. *trackListToJSON*

```
char* trackListToJSON(const List *trackList);
```

This function will convert a list of Tracks - e.g. the tracks list of a `GPXdoc`, or a list returned by `getTracksBetween`- into a JSON string. You can - and should - use `trackToJSON` function defined above.

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

```
[TrackString1,TrackString2,…,TrackStringN]
```

where every `TrackString` is the JSON string returned by `trackToJSON`, and `N` is the number of events in the original list. The order of `TrackString`s must be the same as the order of routes in the original list.

For example, assume that the list contains two Tracks:
- a non-loop track called Mount Steele Trail with the length of 340m would look like this:
- a loop track called Memorial Forest Track with the length of 430m would look like this:

```
[{"name":"Mount Steele Trail","len":340.0,"loop":false},{"name":"Memorial Forest Loop","len":430.0,"loop":true}]
```

Again, 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":"Mount Steele Trail","len":340.0,"loop":false},{"name":"Memorial Forest Loop","len":430.0,"loop":true}]
```

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

5. *GPXtoJSON*

```
char* GPXtoJSON(const GPXdoc* gpx);
```

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

```
{"version":ver,"creator":"crVal","numWaypoints":numW,"numRoutes":numR,"numTracks":numT}
```

where
- `ver` is the GPX version, usually 1.1
- `crVal` is the name of the creator of the GPX file (i.e. value of the `creator` field in the `GPXdoc`)
- `numW` is the number of waypoints in the `GPXdoc`
- `numR` is the number of routes in the `GPXdoc`
- `numT` is the number of tracks in the `GPXdoc`

Notice that the numerical values have no quotes around them.

For example, is we have a GPX doc with
- Version 1.1
- Creator `RunTracker`
- 3 waypoints, 1 route, and 1 track

The output string would be:
```
{"version":1.1,"creator":"RunTracker","numWaypoints":3,"numRoutes":1,"numTracks":1}
```

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

6. Bonus Functions (5%)

The functions below will be necessary in Assignment 3, but you do not have to implement them now.

Please note that the test case(s) for these functions will use all of them together. If any of the tests/functions fail, you will not get the 5% bonus mark. Moreover, the object GPXdoc object created by these functions must pass validation by `validateGPXdoc`.

As a result, do not spend time on these functions unless you have already implemented and tested all of the required Module 1 - 3 functions.

```
void addWaypoint(Route *rt, Waypoint *pt);
```

This function adds a Waypoint struct to the Route struct by inserting it at the end of the waypoints list.

It must not modify the argument `pt` in any way.

If either of the arguments is NULL, the function must do nothing.

```
void addRoute(GPXdoc* doc, Route* rt);
```

This function adds a `Route` struct to the `GPXdoc` struct by inserting it at the end of the routes list.

It must not modify the argument `rt` in any way.

If either of the arguments is NULL, the function must do nothing.

```
GPXdoc* JSONtoGPX(const char* gpxString);
```

This function create a simple `GPXdoc` object from a JSON string. The `gpxString` will have the format:

```
{"version":ver,"creator":"creatorValue"}
```

For example:

`{"version":1.1,"creator":"WebTool"}`

The newly created `GPXdoc` must be property initialized - fields `version`, `creator`, and `namespace` must be filled in (use `http://www.topografix.com/GPX/1/1` for the namespace), and lists `routes`, `tracks`, and `waypoints` must be initialized but empty.

If the argument is NULL, the function must do nothing.

```
Waypoint* JSONtoWaypoint(const char* gpxString);
```

This function will create a simple waypoint from a JSON string. The string will gave the following format:
`{"lat":latVal,"lon":lonVal}`

where `latVal` and `lonVal` are values for the latitude and longitude.

For example: `{"lat":43.537299,"lon":-80.218267}`

The newly created `Waypoint` must be property initialized - fields `latitude` and `longitude` must be filled in, and the list `attributes` must be initialized but empty.

If the argument is NULL, the function must do nothing.

```
Route* JSONtoRoute(const char* gpxString);
```

This function will create an empty named Route from a JSON string. The string will have the following format:
`{"name":"nameVal"}`

Foe example:`{"name":"Reynolds walk"}`

The newly created `Route` must be property initialized - field `name` and must be filled in, and the lists `waypoints` and `attributes` must be initialized but empty.

If the argument is NULL, the function must do nothing.