CIS*2750
Assignment 2
Module 2

You are provided with a temporary header file for A2 Modules 1 and 2 (`GPXParser_A2temp2.h`). 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.

Module 2 functionality

Most of Module 2 functions will need to compete the distance between two waypoints specified using latitudes and longitudes. To compute such a distance use the haversine formula. You can find plenty of examples online, e.g. https://www.movable-type.co.uk/scripts/latlong.html

The link above also allows you to compute the length between any two points, so you can use it to verify your implementation.

If you need to verify the total length of tracks/routes, you can use the tool I mention in A1: http://www.trackreport.net. As always, keep in mind that anything you upload online can be used by anyone for anything, so do not upload GPS data that you don't want random strangers to see (your frequently travelled routes, home location, etc..).

Since GPS positions are intrinsically imprecise - and precision drops under heavy tree canopy, in deep valleys, in some indoor conditions, etc. - we will compare various lengths and locations using specific tolerance. For example, we might consider waypoints located less than 10m from each other to be the same point. As a result, many of the functions below will have an argument for the tolerance.

In addition, if you use real-word GPS-enabled applications, you will notice that the distances are always rounded - usually to the nearest 10m, 100m, or 1km, depending on the overall distance and the purpose of the app. You will implement a very simple rounding function, which will come in handy in Module 3.

Required functions

```
float round10(float len);
```

Function that rounds the length of a track or a route (provided in meters) to the nearest 10 meters. For example:
- round10(403)=400
- round10(404.5)=400
- round10(405)=410
- round10(409)=410

```
float getRouteLen(Route *rt);
```

Function that returns the length of a Route in meters - i.e. the sum of distances between all waypoints that make up the route. The length between any pair of waypoints is computed using the haversine formula. Do not round the result.

Return 0 if the route its NULL

```
float getTrackLen(Track *tr);
```

Function that returns the length of a Track in meters - i.e. the sum of distances between all waypoints that make up the Track. If the track consists of one segment, then the length of the track is the length of the segment. If the track consists of multiple segments, then its length is the length of the segments, plus distances between the adjacent segments.

For example, if a track consists of two segments, its distance is:
length of segment 1 + length of segment 2 + distance between last point in segment 1 and first point in segment 2

Again, do not round the result.

Return 0 if the track is `NULL`

```
int numRoutesWithLength(GPXdoc* doc, float len, float delta);
```

Function that returns the number routes with the specified length using the provided tolerance to compare route lengths.

The length and the tolerance (delta) are provided in meters. If the difference between two lengths is within the tolerance (i.e. <= delta), they are considered equal. For example, given the tolerance of 10m, we will consider the lengths 404m and 411m to be equal to each other.

Return 0 if doc is `NULL`, or if `len` or `delta` are negative

```
int numTracksWithLength(GPXdoc* doc, float len, float delta);
```

Function that returns the number tracks with the specified length using the provided tolerance to compare route lengths.

Return 0 if doc is `NULL`, or if `len` or `delta` are negative

```
bool isLoopRoute(Route* rt, float delta);
```

Function that checks is the specified route is a loop. Return `true` for a loop, `false` otherwise.

For a route to form a loop, it must:
- Have at least 4 waypoints
- Have the distance of less than delta between the first and last points. In other words, the first and last of the route points must be close enough for use to consider them to be roughly the same point, which would create a closed loop.

The delta is, as always, provided in meters.

Return `false` if the `Route` is null or delta is negative

```
bool isLoopTrack(Track *tr, float delta);
```

Function that checks is the specified track is a loop. Return `true` for a loop, `false` otherwise.

For a track to form a loop, it must:
- Have at least 4 waypoints in total in one or more segments
- Have the distance of less than delta between the first and last waypoints. In other words, the first and last points of the track must be close enough for use to consider them to be roughly the same point, which would create a closed loop.

The delta is, as always, provided in meters.

Return `false` if the `Track` is null or delta is negative

Quite often, there is more that one way to get from one location to another. The functions below will allow us to see multiple routes / tracks connecting two locations.

You will need to compare various points to see if a given route / track has start and end points that match the provided source/destination coordinates. Do this by computing the distance between the relevant points, and comparing it to the delta, as you did in the previous functions. If the two points are within the delta of each other, they are considered the same location. For example id delta=10, locations within 9m are considered to be the same one, while locations 11m from each other are not.

```
List* getRoutesBetween(GPXdoc* doc, float sourceLat, float sourceLong,
                       float destLat, float destLong, float delta);
```

Function that returns a list of Route struct pointers, and contains pointers to all Routes connecting the provided source and destination locations.

Make sure that the route list contains **copies** of the Route structs, rather than the references to actual structs in the original `GPXdoc`. This will ensure that the original `GPXdoc` will not be damaged when the user needs to clear/free the list returned by this function.

Return `NULL` if there are no routes between the specified locations, or of the GPXdoc is `NULL`.

```
List* getTracksBetween(GPXdoc* doc, float sourceLat, float sourceLong,
                       float destLat, float destLong, float delta);
```

Function that returns a list of Track struct pointers, and contains pointers to all Tracks connecting the provided source and destination locations.

Make sure that the track list contains **copies** of the Track structs, rather than the references to actual structs in the original `GPXdoc`. This will ensure that the original `GPXdoc` will not be damaged when the user needs to clear/free the list returned by this function.

Return `NULL` if there are no track between the specified locations, or of the GPXdoc is `NULL`.