CIS*2750
Assignment 1
Deadline: Tuesday, October 1, 9:00am
Weight: 15%

Module 1: Primary functions

This is Module 1 of Assignment 1. It will focus on creating a set of structs representing GPX files using `libxml2`. Module 2 will describe a set of accessor and search functions that will accompany the parser and the structs. It will be released after September 20. Module 2 may also contain additional submission details.

Description

In this assignment, you need to implement a library to parse the GPX files. GPX is one of the most common formats for representing a collection of waypoints on a map. If you ever went running and tracked your run using an app, it most likely used GPX to store your route.

The link to the format description is posted in the Assignment 1 description. Make sure you understand the format before doing the assignment. In fact, it's a good idea to read through it before your read the rest of this document.

According to the specification, a GPX object contains:
- a `gpx` element, with some metadata. We will not need to parse this metadata.
- 0 or more waypoints, routes, and tracks.
- routes and tracks are made up of ordered lists of points
- points, routs, and tracks can have meta-data - names, dates, etc..

GPX is a fairly simple format, and our Assignment 1 the parser will be fairly complete, other than our handling of XML namespaces (for simplicity) and `gpx` metadata.

This structure is represented by the various types in `GPXParser.h`.

Notes:
- What we call an `Attribute` in this assignment isn't technically an attribute in the XML sense. For example, the GPX `wptType` (used to represents various types of points) only has two XML attributes: `lat` and `lon`. However, it can have a number of children - `ele` (elevation), `time`, `name`, etc. - that provide additional information. We refer to these children as "attributes", and the `Attribute` type is meant to represent them. So for example, if inside a waypoint (`wpt` element), you find a child `<ele>334.0</ele>`, you will create an Attribute with the name `ele` and value `334.0`. This attribute will be added to that waypoint's `attributes` list
- Names of tracks, waypoints, and routes, if present, must be placed in the `name` field of the appropriate struct. They will be particularly useful in our project, and we want to make accessing them easy. Do not place names in the `attributes` list.

The GPX standard uses the common XML language. To help you build the parser, you will use the `libxml2` library, once of the most common C XML parsers. It will build an XML tree for you from an GPX file. You will then use it to create an `GPXdoc` struct with all of its components. The XML parser will do most of the heavy lifting for you.

The documentation on libxml2 is available here: http://www.xmlsoft.org/html/index.html. You can use the sample code to get started, but you **must** site it in your submission:
- in the header of our source file that uses this code, clearly state which sample code from xmlsoft.org you've used
- include links to each of these source files, e.g. http://www.xmlsoft.org/examples/tree1.c.

You cannot use any other sample code.

Your assignment will be graded using an automated test suite, so you must follow all requirements exactly, or you will lose marks.

Required Functions

Read the comments in `GPXParser.h` carefully. They provide additional implementation details. You **must** implement **every** function in `GPXParser.h`; they are also listed below. If you do not complete any of the functions, you **must** provide a stub for every one them.

Applications using the parser library will include `GPXParser.h` in their main program. The `GPXParser.h` header has been provided for you. Do not change it in any way. `GPXParser.h` is the public "face" of our library API. All the helper functions are internal implementation details, and should not be publicly/globally visible. When we grade your code, we will use the standard `GPXParser.h` to compile and run it.

If you create additional header files, include them in the `.c` files that use them.

GPX parser functions

```
GPXdoc* createGPXdoc(char* fileName)
```

This function does the parsing and allocates a `GPXdoc` object. It accepts a filename. If the file has been parsed successfully, a pointer to the `GPXdoc` object is returned. If the parsing fails for any reason, the function must return `NULL`.

Parsing can fail for a number of reasons, but libxml hides them from us. The `xmlDocGetRootElement` function will simply return NULL instead of an XML tree if the file is invalid for any reason.

```
char* GPXdocToString(GPXdoc* doc)
```

This function returns a humanly readable string representation of the entire GPX document. It will be used mostly by you, for debugging your parser. It must not modify the document in any way. The function must allocate the string dynamically.

```
void deleteGPXdoc(GPXdoc* doc);
```

This function deallocates the object, including all of its subcomponents.

Helper functions

In addition the above functions, you must also write a number of helper functions. We will need to store the types `Attribute`, `Track`, `TrackSegment`, `Route`, and `Waypoint` in lists.

```
void deleteAttribute( void* data);
char* attributeToString( void* data);
int compareAttributes(const void *first, const void *second);

void deleteWaypoint(void* data);
char* waypointToString( void* data);
int compareWaypoints(const void *first, const void *second);

void deleteRoute(void* data);
char* routeToString(void* data);
int compareRoutes(const void *first, const void *second);

void deleteTrackSegment(void* data);
char* trackSegmentToString(void* data);
int compareTrackSegments(const void *first, const void *second);
```

```
void deleteTrack(void* data);
char* trackToString(void* data);
int compareTracks(const void *first, const void *second);
```

Additional guidelines and requirements

In addition, it is strongly recommended that you write additional helpers functions for parsing the file - e.g. creating a `Waypoint`, `Route`, etc. from an `xmlNode`.

You are free to create your additional "helper functions" in a separate `.c` file, if you find some recurring processing steps that you would like to factor out into a single place. Do **not** place headers for these additional helper function in `GPXParser.h`. They must be in a separate header file, since they are internal to your implementation and not for public users of the utility package.

For your own test purposes, you will also want to code a main program in another `.c` file that calls your functions with a variety of test cases, However, you **must not** submit that program. Also, do **not** put your main() function in `GPXParser.h`. Failure to do this may cause the test program will fail if you incorrectly include main() in our shared library file; **you will lose marks for that**, and may get a 0 for the assignment.

Your functions are supposed to be robust. They will be tested with various kinds of invalid data and must detect problems without crashing. If your functions encounter a problem, they must free all memory and return.

Function naming

You are welcome to name your helper functions as you see fit. However, **do not** put the underscore (_) character at the start of your function names. That is reserved solely for the test harness functions. Failure to do so may result in run-time errors due to name collisions - and a grade of zero as a result.

Linked list

You are expected to use a linked list for storing various GPX components. You can use the list implementation that has been posted on Moodle. You can also use your own. Whatever you do, your implementation **must** be compliant with the List API defined in `LinkedListAPI.h`. Failure to do so may result in a grade deductions, up to and including a grade of zero.

Recommended development path:

1. Implement a simple XML parser
2. Implement a simple `GPXdoc` parser that that extracts the required property of the GPX file (an `gpx` component and a namespace). You can skip the namespace for now if it gives you trouble - just remember to add it later.
3. Add a basic `deleteGPXdoc` functionality and test for memory leaks
4. Add a basic `GPXdocToString` functionality and test for memory leaks
5. Add handling of `Waypoints` for a GPX file
    1. Update `deleteGPXdoc` and `GPXdocToString` functionality.
    2. Test for memory leaks
6. Add handling of `Attributes` for `Waypoints`
    1. Update `deleteGPXdoc` and `GPXdocToString` functionality.
    2. Test for memory leaks
7. Add handling of `Routes` with attributes
    1. ...
    2. ...

8. Add handling of `Tracks` with attributes
    1. Have a beer and ignore the rest of the assignment.
    2. Just kidding. Yes, keep updating `delete`/`toString` functions, and testing for leaks
9. Implement the get*** functions (Module 2)
10. Test for memory leaks

Important points

Do:

- **Do** be careful about upper/lower case.
- **Do** include comments with your name and student ID at the top of every file you submit

Do not:

- **Do not** submit `GPXParser.h` or `LinkedListAPI.h`. If they are submitted, they will be overwritten.
- **Do not** change the given typedefs or function prototypes in `GPXParser.h`
- **Do not** hardcode any path or directory information into #include statements, e.g. `#include "../include/SomeHeader.h`
- **Do not** submit any `main()` functions
- **Do not** use `exit()` function calls anywhere in your code. Since you're writing a library, it must always return the control to the caller using the `return` statement
- **Do not** exit the program from one of the parser functions if a problem is encountered, return an error value instead.
- **Do not** print anything to the command line.
- **Do not** assume that your pointers are valid. Always check for NULL pointers in function arguments.

Failure to follow any of the above points may result in loss of marks, or even a zero for the assignment if they cause compiler errors with the test harness.

Submission structure

The submission must have the following directory structure:

`assign1/`	- contains the `Makefile` file.
`assign1/bin`	- should be empty, but this is where the `Makefile` will place `libgpxparse.so`.
`assign1/src`	- contains `GPXParser.c` , `LinkedListAPI.c`, and your additional source files.
`assign1/include`	- contains your additional headers. Do not submit `GPXParser.h` and `LinkedListAPI.h`.

Makefile

You will need to provide a `Makefile` with the following functionality:

- `make parser` creates a shared library `libgpxparse.so` in `assign1/bin`
- `make clean` removes all `.o` and `.so` files

Evaluation

Your code will be tested by an automated harness. You will submit a Makefile, which will be used to produce a shared library. This library will then be tested on the SoCS Linux servers by a precompiled executable file containing the test harness, as well as another executable file with simple memory leak tests. Your library must implement the assignment API exactly as specified, or you will get run-time errors because the executable files will not find functions in the library that they expect.

Your code must compile, run, and have all of the specified functionality implemented. Any compiler errors, as well as any failure to run with the automated test harness, will result in the automatic grade of **zero** for the assignment. Infinite loops will also result in a grade of **zero**.

Marks will be deducted for:
- Incorrect and missing functionality
- Deviations from the assignment requirements
- Run-time errors, including infinite loops
- Compiler warnings
- Memory leaks reported by valgrind
- Memory errors reported by valgrind
- Failure to follow submission instructions

Submission

Submit your files as a Zip archive using Moodle. File name must be `A1FirstnameLastname.zip`.

Late submissions: see course outline for late submission policies.

This assignment is individual work and is subject to the University Academic Misconduct Policy. See course outline for details)