CIS*2750
Assignment 1
Deadline: Tuesday, February 1, 9:00am
Weight: 25%

Module 1: Primary functions

This is Module 1 of Assignment 1. It will focus on creating a set of structs representing SVG 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 January 15. Module 2 may also contain additional submission details.

Description

In this assignment, you need to implement a library to parse the SVG files. SVG is one of the most common vector graphics formats, and is supported by every Web browser.

The link to the format description is posted in the Assignment 1 description. Make sure you understand the format before doing the assignment.

According to the specification, an SVG object contains:
- one or more `svg` elements
- 0 or more groups, which can contain other groups
- a large number of primitives, along with text, colour attributes, fill attributes, etc..

Our Assignment 1 parser will assume a somewhat simpler SVG file:
- one `svg` element
- 0 or more groups, which can contain other groups
- rectangles, circles, and paths - no other geometric primitives

Read the specification and pay attention to the specification details, e.g. what to do if the attributes x or y are not specified in a circle or rectangle.

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

The SVG 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 SVG file. You will then use it to create an `SVG` 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 provided there 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 can also use the sample code I have provided for you in the A1 description on the course website and discussed in Lecture 2. 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 `SVGParser.h` carefully. They provide additional implementation details. You **must** implement **every** function in `SVGParser.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 `SVGParser.h` in their main program. The `SVGParser.h` header has been provided for you. Do not change it in any way. `SVGParser.h` is the public "face" of our vector image 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 `SVGParser.h` to compile and run it.

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

SVG parser functions

```
SVG* createSVG(const char* fileName);
```

This function does the parsing and allocates an SVG struct. It accepts a filename, which must not be modified in any way. If the file has been parsed successfully, a pointer to the newly created `SVG` struct 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 `xmlReadFile` function will simply return NULL instead of an XML doc if the file is invalid for any reason.

```
char* SVGToString(const SVG* img);
```

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

```
void deleteSVG(SVG* img);
```

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 `Group`, `Rectangle`, `Circle`, `Path`, and `Attribute` in lists.

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

void deleteGroup(void* data);
char* groupToString( void* data);
int compareGroups(const void *first, const void *second);

void deleteRectangle(void* data);
char* rectangleToString(void* data);
int compareRectangles(const void *first, const void *second);

void deleteCircle(void* data);
char* circleToString(void* data);
int compareCircles(const void *first, const void *second);
```

```
void deletePath(void* data);
char* pathToString(void* data);
int comparePaths(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 `Circle`, `Group`, etc. from an `xmlNode`. You should also write delete...() functions for all the structs, since they will all be dynamically allocated.

All required functions **must** be placed in files prefaced with `SVG` - e.g. `SVGParser.c`, `SVGHelpers.c`, etc.. 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 `SVGParser.h`. They must be in a separate header file, since they are internal to your implementation and not for public users of the parser package. This separate header file also **must** be prefaced with `SVG`, e.g. `SVGHelpers.c`.

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, but you won't submit that program. The file containing the main function **must not** have the SVG prefix.

Only the files with the `SVG` prefix will be pulled from Git and used for grading. Do **not** put your main() function into any of the files with the `SVG` prefix. Otherwise, the test executable will fail due to multiple definitions of main(); **you will lose marks for that**, and may get a **zero** 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 (0) as a result.

Linked list

You are expected to use a linked list for storing various vector image components. You can use the list implementation that I use in the class examples (and that you use for A0). You can also use your own. However, 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 `SVG` parser that that extracts the required property of the SVG image (an `svg` component and a namespace).
3. Add a basic `deleteSVG` functionality and test for memory leaks
4. Add a basic `SVGToString` functionality and test for memory leaks

5. Add handling of multiple attributes for an `svg` component
   1. Update `deleteSVG` and `SVGToString` functionality.
   2. Test for memory leaks
6. Add handling of one of the geometric primitive types, e.g. `Rectangles`.
   1. Update `deleteSVG` and `SVGToString` functionality.
   2. Test for memory leaks
7. Add handling of `Rectangles` with attributes

8. Add handling of `Circles` and `Paths` (one at a time)
   1. ...
   2. ...
9. Add handling of `Circles` and `Paths` with attributes
   1. ...
   2. ...
10. Add handling of `Groups`
    1. you guessed it
    2. you guessed it
11. Add handling of `Groups` 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
12. Implement proper the Module 2 functions
13. 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 use the `SVG` prefix for all .c and .h files that you wan to include in your graded A1

Do not:

- **Do not** submit `SVGParser.h` or `LinkedListAPI.h`. We will the standard versions of these headers when grading you - not the ones that you submit.
- **Do not** change the given typedefs or function prototypes in `SVGParser.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, and make sure your functions handle the NULL arguments correctly.

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 details on the submission structure will be posted once your repos have been created, prepopulated, and tested.

Evaluation

Your code will be tested by an automated harness. Your code will be compiled using a standard Makefile, provided for your reference in the Assignment 1 folder in the course website. 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 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

You must commit and push your code to the appropriate repo on gitlab.socs.uoguelph.ca by the specified date. The details on the submission process will be posted once your repos have been created and tested. **No other submissions will be accepted.**

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)