Preliminary harness Instructions

Instructions

Use the Makefile provided in the A1 repo (or the online assignment description) to create a shared library file `libsvgparser.so` and place it in the same directory as the `test1pre` executable. Remember to use the original, unmodified versions of `LinkedListAPI.h` and `SVGParser.h` - they have been provided for your reference in the assignment description.

To test your code, run the executable `test1pre`. You should see a long list of test cases, with the max displayed score of 54. The executable was compiled on the SoCS Linux server (linux.socs.uoguelph.ca), so that is where you must run it. Needless to say, the library must be compiled there as well.

Troubleshooting harness execution

- Make sure the execute permission set for `test1pre`, which you can do with the following command: `chmod 755 test1pre.` If you don't do this, you will get an error:
  `./test1pre: Permission denied`

- Make sure that `libsvgparser.so` is in the same directory as `test1pre`, and your `LD_LIBRARY_PATH` variable is configured correctly, as discussed in class (e.g. January 25 lecture). If it is not, you will get an error:
  `./test1pre: error while loading shared libraries: libsvgparser.so: cannot open shared object file: No such file or directory`

  The error is caused by the loader (Lecture 3a) not being able to find the path to the shared library `libsvgparser.so`. The solution is to add the current directory to the set paths that the loader searches:
  `export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:.`

If you want to see which shared libraries an executable uses - and whether the loader can find them - you can type `ldd fileName`. In our case, if we don't set `LD_LIBRARY_PATH`, typing `ldd test1pre` will show:

```
linux-vdso.so.1 (0x00007fff9b974000)
libsvgparser.so => not found
libm.so.6 => /lib/x86_64-linux-gnu/libm.so.6 (0x00007fc1238cd000)
libpthread.so.0 => /lib/x86_64-linux-gnu/libpthread.so.0 (0x00007fc1238ac000)
libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0x00007fc1236eb000)
/lib64/ld-linux-x86-64.so.2 (0x00007fc123a78000)
```

As you can see, the loader cannot resolve the shared library dependency `libsvgparser.so`.

Once `LD_LIBRARY_PATH` is set, you will see:

```
linux-vdso.so.1 (0x00007ffebb962000)
libsvgparser.so (0x00007f77889e9000)
libm.so.6 => /lib/x86_64-linux-gnu/libm.so.6 (0x00007f778884e000)
libpthread.so.0 => /lib/x86_64-linux-gnu/libpthread.so.0 (0x00007f778882d000)
libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0x00007f778866c000)
/lib64/ld-linux-x86-64.so.2 (0x00007f7788a02000)
```

The loader can now resolve the shared library dependency `libsvgparser.so`, and the harness will run.

Harness details

The test harness works as follows: for each specific file it creates a `SVG` using your `createSVG` function. It then executes various functions and compares the result it receives to the expected result.

For example, a test case might include `numRectsWithArea`:

- The test would create an `SVG` struct from a valid .svg file using your `createSVG`.
- If your `createSVG` returns NULL when called on a valid file, the test would fail.
- The test would then call `getRects` using the new `SVG` struct at the argument.
- If the number returned by `numRectsWithArea` matches the expected correct rectangle count, the test would pass. Otherwise, it would fail.

Additional notes:

- Some tests contain multiple sub-tests. If your code fails any of the subtests, it fails the entire test block. The score for that test block is then 0.
- If your code crashes or enters an infinite loop for some test case, it fails that test case.
- Some tests will pass either non-existent files or files with invalid XML structure to your `createSVG`. Your `createSVG` is expected to return NULL in those cases, as stated in the A1 specification.
- All tests with valid files will pass a name of a valid .svg file to your `createSVG`.
- Some tests will include arguments for which the expected count is zero, or the returned value is null. For example:
  - Subtest 1.3 uses your `getRects` on a `SVG` created by your `createSVG` from Emoji_poo.svg. Since that file has no rectangles, the test case expects your `getRects` to return NULL.
  - Subtest 7.1 uses your `numPathsWithdata` to search for a specific in an `SVG` created by your `createSVG` from rect.svg. Since that file has no paths, the test case expects your `numPathsWithdata` to return 0.
- Some tests will pass NULL arguments to your code. Your code is expected to verify that the argument pointers are not NULL, and it must not crash.

Using the harness

Please note that the test harness is a testing tool and not a debugging tool. It will not tell why your code fails a test - it simply tells that the output of your function does not match the expected output - and therefore does not match the assignment specification.

You are responsible for finding and fixing errors in your code. Any differences between expected output and what is returned by your code would cause the code to fail. This would include mismatched strings, mismatched list lengths, etc..

Some suggestions on debugging:
- Start with the simplest files.
- Verify that the values for each field are correct. Examine each string in your SVG vey carefully, including the string length.
- Verify that each list has the correct length. Make sure you don't have any duplicate or unnecessary attributes, sub-groups, etc..
- Make sure that the lists use the correct `detele...`, `compare...`, and `...toString` functions.
- Make sure all lists are initialized!
- Make sure that all string that should be empty are indeed empty - and not just uninitialized.
- Make sure all dynamically-allocated elements have been allocated correctly.

Test Harness and Memory

The test harness never tries to deallocate memory, apart from the tests for the `deleteSVG` function. This is done to minimize the chances of your code crashing during the tests. As a result, it will definitely leak a lot of memory, so **do not** try running it through `valgrind` in an effort to find memory bugs in your code. You will need to write your own tests for finding memory leaks and errors.

Because of this, the test harness does not include any tests for memory leaks or memory errors. Memory testing will be done by a different test harness. Strategies for testing your code for memory issues are going to be discussed in c lass.