Preliminary harness Instructions

Instructions

Get the correct version:
- If you are using a computer with an Intel-based CPU, download the file `test1preIntel.zip`
- If you are using a computer with an ARM-based CPU - e.g. a Mac with an M1 - M4 chip - download the file `test1preARM.zip`

If you are not sure, check the specifications of your computer online.

Use your Makefile to create a shared library file `libvcparser.so`, as per Assignment 1 instructions, and place it in the same directory as the `test1pre` executable. Remember to use the original, unmodified versions of `LinkedListAPI.h` and `VCParser.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 51. The executables were compiled using the CIS*2750 Docker container on Intel- and ARM-based computers for you, so you must run this executable in the CIS*2750 Docker container. Needless to say, your library file 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 `libvcparser.so` is in the same directory as `test1pre`, and your `LD_LIBRARY_PATH` variable is configured correctly, as discussed at the end of Lecture 2a. If it is not, you will get an error:
  `./test1pre: error while loading shared libraries: libvcparser.so: cannot open shared object file: No such file or directory`

  The error is caused by the loader (Lecture 2a) not being able to find the path to the shared library `libvcparser.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 something like this:

```
linux-vdso.so.1 (0x00007fff9b974000)
libvcparser.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 `libvcparser.so`.

Once `LD_LIBRARY_PATH` is set, you will see something like this:

```
linux-vdso.so.1 (0x00007ffebb962000)
libvcparser.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 `libvcparser.so`, and the harness will run. The output for the code that passes all tests will look like this:

```
Test 1 (2%): Testing createCard. Creating vCard object from a valid file (testFiles/valid/testCardMin.vcf): PASSED 1/1 tests
    SUCCESS: Subtest 1.1: file testFiles/valid/testCardMin.vcf parsed correctly

Test 2 (3%): Testing createCard. Creating vCard object from a valid file (testFiles/valid/testCardProp-simpleVal.vcf): PASSED 1/1 tests
    SUCCESS: Subtest 2.1: file testFiles/valid/testCardProp-simpleVal.vcf parsed correctly

Test 3 (3%): Testing createCard. Creating vCard object from a valid file (testFiles/valid/testCardProps-simpleVal.vcf): PASSED 1/1 tests
    SUCCESS: Subtest 3.1: file testFiles/valid/testCardProps-simpleVal.vcf parsed correctly

Test 4 (3%): Testing createCard. Creating vCard object from a valid file (testFiles/valid/testCardN-compVal.vcf): PASSED 1/1 tests
    SUCCESS: Subtest 4.1: file testFiles/valid/testCardN-compVal.vcf parsed correctly

Test 5 (3%): Testing createCard. Creating vCard object from a valid file (testFiles/valid/testCardProp-Param.vcf): PASSED 1/1 tests
    SUCCESS: Subtest 5.1: file testFiles/valid/testCardProp-Param.vcf parsed correctly

Test 6 (4%): Testing createCard. Creating vCard object from a valid file (testFiles/valid/testCard-SimpleFold1.vcf): PASSED 1/1 tests
    SUCCESS: Subtest 6.1: file testFiles/valid/testCard-SimpleFold1.vcf parsed correctly

Test 7 (3%): Testing createCard. Creating vCard object from a valid file (testFiles/valid/testCard-Ann.vcf): PASSED 1/1 tests
    SUCCESS: Subtest 7.1: file testFiles/valid/testCard-Ann.vcf parsed correctly

Test 8 (3%): Testing createCard. Creating vCard object from a valid file (testFiles/valid/testCard-TruncBday.vcf): PASSED 1/1 tests
    SUCCESS: Subtest 8.1: file testFiles/valid/testCard-TruncBday.vcf parsed correctly

Test 9 (3%): Testing createCard. Creating vCard object from a valid file (testFiles/valid/testCard-BdayTime.vcf): PASSED 1/1 tests
    SUCCESS: Subtest 9.1: file testFiles/valid/testCard-BdayTime.vcf parsed correctly

Test 10 (3%): Testing createCard. Creating vCard object from a valid file (testFiles/valid/testCard-BdayText.vcf): PASSED 1/1 tests
    SUCCESS: Subtest 10.1: file testFiles/valid/testCard-BdayText.vcf parsed correctly

Test 11 (3%): Testing createCard. Creating vCard object from a valid file (testFiles/valid/testCardProps-Groups.vcf): PASSED 1/1 tests
    SUCCESS: Subtest 11.1: file testFiles/valid/testCardProps-Groups.vcf parsed correctly

Test 12 (2%): Creating a vCard object from invalid files: PASSED 1/1 tests
    SUCCESS: Subtest 12.1: Reading a NULL file name.

Test 13 (5%): Creating a vCard object from files with invalid Cards: PASSED 2/2 tests
    SUCCESS: Subtest 13.1: Reading a file with non-CRLF line endings (testFiles/invCard/testCardInvEndings.vcf)
    SUCCESS: Subtest 13.2: Reading a file with a missing END tag (testFiles/invCard/testCardNoEnd.vcf)

Test 14 (5%): Creating a vCard object from files with invalid Properties: PASSED 2/2 tests
    SUCCESS: Subtest 14.1: Reading a file with a missing property name (testFiles/invProp/testCardNoPropName.vcf)
    SUCCESS: Subtest 14.2: Reading a file with an invalid parameter (testFiles/invProp/testCardPropInvParam2.vcf)

Test 15 (2%): Testing errorToString: PASSED 1/1 tests
    SUCCESS: Subtest 15.1: errorToString test

Test 16 (2%): Testing cardToString: PASSED 2/2 tests
    SUCCESS: Subtest 16.1: Printed a vallid non-NULL reference Card object.
    SUCCESS: Subtest 16.2: cardToString handled NULL Card object correctly.

Test 17 (2%): Testing deleteCard on objects created from valid files.  NOTE: see valgrind output for deletion memory leak report: PASSED 2/2 tests
    SUCCESS: Subtest 17.1: Card object created from file testFiles/valid/testCardMin.vcf deleted with no crashes
    SUCCESS: Subtest 17.2: Reference card object deleted with no crashes
Score: 51
```

If your code fails some test cases, the test harness score will be below 51 and the failed tests will get the appropriate feedback (and will be highlighted in red).

Harness details

The test harness works as follows: for each specific file it compares the contents of the `Card` struct returned by your `createCard` to a reference `Card` created manually. If each value matches, the test passes. If there are any differences, the test fails.

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. Note that the difference that would cause the code to fail would include mismatched strings, mismatched list lengths, etc..

Some suggestions:

- Start with the simplest files.
- Make sure you have no stray CR and LF (`\r` and `\n`) characters stuck in the strings. Remember that neither CR not LF is visible. This is particularly likely culprit if your code fails every single `createCard` test.
- Verify that the values for each field are correct. Examine each string in your Card vey carefully, including the string length. This can help catch the stray CR or LF characters.
- Verify that each list has the correct length. Make sure you don't have any duplicate or unnecessary properties, parameters, or values.

The test harness never tries to deallocate memory - apart from the tests for the `deleteCard` function - 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 create your own tests for 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 different software.