# CPP string assignment report

The assignment was to implement a STL style container for strings in C++ and test it with a custom test driver. My implementation of the string class uses internally null terminated character arrays to represent strings.

The size of the array is always just enough so that the array and the null character can fit. It only occurred to me at the end of the project that this was quite a poor design. I should have done it so that the array size could have been bigger than the actual string and it would have been resized only when needed by for example doubling the size. While this would have been much more efficient design, I didn’t change it since I think the realization in this case was enough for my learning, implementing it would have been trivial.

There’s also another thing I would change in my design. The myString class currently has a variable called sz, which is equal to the number of the characters in the string. Often in the code I have to use sz + 1 because the array has to hold also the null character. The code would have maybe been more readable if size would have been one bigger than the number of characters and the size function would have just returned sz - 1.

In the beginning of the project I had even problems with resizing the array since I was not familiar with C++ before. I don’t know whether my solution to copy the array’s contents to a temporary array and then resize the original is the best one, but it seems to work. I also tried to keep the class const correct and use exceptions to keep the string’s state valid in most of the situations. My implementation also tries to achieve safety over efficiency, so for example the resize function makes a backup of the string’s contents and resets it if an exception occurs.

All the classes have separate header and cpp files as was requested. The iterator class declaration is, however, defined in myString.h because I thought it logically belongs there. I had some problems with including the files, they were giving many errors. I solved it by using include guards in the header files.

Testing is divided into two separate classes. The tests class includes all the tests and the test driver class executes the tests and also provides some static functions (e.g. assertEquals) that the test class is supposed to use in all the tests to check that the outputs of the tested functions are valid. The test class itself saves all of the tests in a map which the testdriver gets from it and subsequently calls all the test functions in a loop. The testdriver then prints out the results of the tests on the console.

The program can be compiled either by the provided makefile (run command make in the folder where the files are) or by running command: g++ -std=c++11 -Wall -o stringtest.exe main.cpp myString.cpp tests.cpp testdriver.cpp iterator.cpp –DNDEBUG, which is the one the makefile runs anyway.