# CPP string assignment report

The assignment was to implement an 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.

In my implementation 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. I didn’t actually have the null character in the beginning at all, but I had some errors that were solved by adding it.

In the beginning of the project I had even problems with resizing the array since this was my very first time coding C++. 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 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. I tried to keep the header files clear by having the public functions in the beginning and the private members last. I also tried to avoid defining functions in the header files, but in the test driver’s header I had to define some template functions, because I couldn’t get them working otherwise. I also read online that it is okay to define template functions in the header.

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 tested functions behave as expected. The test class itself saves all of the tests in a map which the test driver gets from it and subsequently calls all the test functions in a loop. I think in Java it would be possible to call all the test methods by reflection, but since I couldn’t find any similar functionality in C++, this seemed to be a fairly good solution. After executing the functions the test driver then prints out the results of the tests on the console as well as the thrown exceptions if there are any.

The test driver is sufficient for this purpose, but of course it is not very general purpose. Also one problem with it is that the Tests class’ test methods have to themselves inspect the Booleans returned by the assertEquals functions and keep a record of whether any assert failed. This is because at the end of the test the function is required to return a Boolean to the test driver. The driver inspects the Boolean and infers from that whether the test failed or not. The driver is also tightly coupled to the console: it does not have any buffers, but instead the assertEquals functions and the run function print results directly to 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.