Character array

You are given the abstract data type (ADT) *CharacterArray*.
The class should contain a string in a dynamic ***array***.
A dynamic array of characters is declared as a *pointer*, and then instantiated with the ***new*** command and the ***[]*** operator:

```
char* arr;
arr = new char[NUMBER_OF_CHARACTERS_IN_ARRAY];
```

After that it can be used as a regular array of characters.
Note that you must make sure to have **'\0'** at the end of the string at all times, behind the last "real" element. Thus, at the beginning the first element of the array is **'\0'**. This makes it a "null terminated" string, so that when it is printed the stream knows when to stop printing characters (as soon as it finds **'\0'**).
You are to implement all the functions in the class so that the declared functions work as described below:

- ***CharacterArray()***
  A constructor with no parameters. It must build an empty array with the capacity `INITIAL_CAPACITY`. Remember to also have room for the **'\0'** at the end. Here you must also initialize all other variables that you have declared in your class, for example size and capacity.
- ***CharacterArray(char* str, int length)***
  A constructor with parameters. It must build an array with enough capacity to hold a string of length ***length***. It must then copy the first ***length*** characters from the character array ***string***. Remember to also have room for the **'\0'** at the end. Here you must also initialize all other variables that you have declared in your class, for example size and capacity. They must be initialized so that they are consistent with the parameters.
- ***~CharacterArray()***
  The deconstructor. It must delete all memory that has been allocated by the class. Mainly it must delete the character array. Remember that the syntax `delete[] arr` deletes memory for the whole dynamic array, rather than a single instance.
- ***void append(char c)***
  The function adds one character with the value ***c*** at the end of the array.
- ***void insert(char c, int index)***
  The function adds one ***char*** into the array at the location ***index***. If ***index*** is 0 you add at the beginning of the string, if ***index*** equals the length of the string, add at the end (same as append. For a string of length 3 there are four possible indexes to add at, 0, 1, 2, 3.
  Throw an ***IndexOutOfBoundsException*** *if* ***index*** *is outside that interval.*
- ***void setAt(char c, int index)***
  The function sets the value of the character at the location ***index***. Only values that are already there can be set. For a string of length 3 there are three possible indexes to set the value at, 0, 1, 2.
  Throw an ***IndexOutOfBoundsException*** *if* ***index*** *is outside that interval.*

- ***char getAt(int index)***
  The function returns the value of the character at the location ***index***. Only values that are already there can be returned. For a string of length 3 there are three possible indexes to get the value at, 0, 1, 2.
  *Throw an **IndexOutOfBoundsException** if **index** is outside that interval.*
- ***char pop_back()***
  The function returns the character at the end of the array (last usable index), and removes that element from the array. This is simply done by shortening the effective array by one. Remember to move the **'\0'** at the end to the correct location.
  *Throw an **EmptyException** if there is no element to return. This check and throw has already been implemented and you can use this as a template for throwing exceptions in other functions.*
- ***char removeAt(int index)***
  The function returns the character at the location ***index***, and removes that element from the array. In order to do this the whole array, behind ***index***, must be moved forward by one. Then you must shorten the effective array by one and remember to move the **'\0'** at the end to the correct location.
  Only values that are already there can be returned and removed. For a string of length 3 there are three possible indexes to remove the value at, 0, 1, 2.
  *Throw an **IndexOutOfBoundsException** if **index** is outside that interval.*
- ***int clear()***
  Empties the effective array. Make your variables consistent so that the ***array size*** will be **0** and the ***operator <<*** will write an empty string ***""*** to the stream.
- ***int length()***
  Returns the amount of characters that have been added to the array (*not* including **'\0'**).
- ***bool isEmpty()***
  Returns true if the effective array is empty (array size 0), otherwise false.
- ***char* substring(int startIndex, int length)***
  The function returns a *partial string*, a new array of ***char*** where the first element is the element at ***startIndex*** in your array and after that come the next characters in your array, so that the substring`s length is ***length***. Behind the last element you must put **'\0'** so you must allocate memory for the new array accordingly. *Throw an **IndexOutOfBoundsException** if the beginning or end of the substring reaches outside your array.*
- ***Overridden stream operator <<***
  Writes the contents of the string to the stream. Make sure to do it in the least complex way possible.

You must implement this data structure using an array of characters (***char***) and there must not be any limit on how many characters could be added to it. The tests obviously only add a finite amount of data to it, but your implementation should have no predefined limit. This means that when the capacity of your array (the memory allocated for it using ***new***) is full or equals the size of your effective array (the amount of characters that have been added already) you must

reallocate memory for it. This is done by allocating memory for a new char array (you can use a temp ***char****), ***double*** the size of the original one, and then moving all the data from the original array into the new one. Finally you must **delete** the memory allocated for the original array and make sure your main array pointer (char* arr) *points to the new array location*. Try to do this in a separate function and call it from any place where this could be needed (where the array might grow).
The abstract data type usable to an outside user (the main function in this case) should continue to be limited to the functions originally on the ADT. This means that all class variables and helper functions should be declared as private except the functions that are already declared public and are part of the ADT.

The file **ExampleOutput.txt** shows the output from a run that uses all these functions.

The assignment will be returned through Mooshak.

- You get ***three attempts*** to enter it into Mooshak. The third one is final.
- Send a ZIP folder into Mooshak containing characterarray.h and characterarray.cpp.

You must make really good tests in your own main function.

- The current main function contains helper functions and examples of their use.
- You must make sure that you test all edge cases.
    - remove first element
    - remove last element
    - remove at index one higher than last element to test exception in edge case
- Be absolutely sure that your code works and that you understand why it works before trying to understand Mooshak's error messages.
- Make sure your program doesn't leak memory.
    - For every new there should be a delete.
    - Deconstructor should delete everything that was allocated.