Documentation of String-"Class"

Non-member functions:

`String createString(void)`
returns a `String` that is to be used to initialize an uninitialized `String`. Remember to destroy the `String` using its member function `void destructor(String *)` after it is no longer needed.

Non-member typedefs:

`string_size_type` The type that represents the size of the `String`.

`string_value_type` The type that represents the type of the elements contained by the `String`.

Public member functions:

`void destructor(String *)`
Destroys the `String` that the parameter points to.

`void readFrom(String *, FILE *)`
Reads from the stream passed into the `FILE *` parameter until `EOF` and writes it into the `String` that the first parameter points to.

`void writeTo(String const *, FILE *)`
Writes the content of the `String` pointed to by the first parameter to the stream passed into the `FILE *` parameter.

`string_size_type size(String const *)`
Returns the size of the `String` pointed to by the parameter. The size is the amount of `string_value_types` in the `String` that are not equal to '\0' until the first '\0' `string_value_type`. This is equivalent to `strlen` for `char const *`.

`string_size_type capacity(String const *)`
Returns the size of the largest string that the `String` pointed to by the parameter can hold in its current state. The size of a string shall be defined as it is returned by `strlen`. Thus a string of size 5 (that is actually 6 bytes large, because of the '\0' character) will fit into a `String` of size 5, as the `String` is always 1 byte larger than what `capacity` returns to be able to hold that '\0' character.

`string_value_type *data(String const *)`

Returns a pointer to the beginning of the internal buffer of the `String` that the parameter points to. This can be used to use the typical C-style string functions defined in `string.h`. Carefulness is advised.

`string_value_type *at(String const *, string_size_type)`
Returns a pointer to the `string_value_type` at the index passed into the second parameter in the `String` pointed to by the first parameter. If compiled in debug mode (`NDEBUG` is not defined) this function performs range checks, otherwise (if compiled in release mode (`NDEBUG` defined)) the `String` may be accessed out of bounds which may or may not crash the application with a segmentation fault / access violation.

`string_value_type *front(String const *)`
Returns a pointer to the first `string_value_type` of the `String` pointed to by the parameter.

`string_value_type *back(String const *)`
Returns a pointer to the last `string_value_type` of the `String` pointed to by the parameter. That is the `string_value_type` before the first '\0'.

`void clear(String *)`
Clears the `String` pointed to by the parameter so that it is empty, that is all of its buffers bytes are zeroed out so that a call to `size` will return 0.

`bool isEmpty(String const *)`
Returns a non-zero value if the `String` pointed to by the first parameter has a `size` of 0. Otherwise a value of 0 is returned.

`void shrinkToFit(String *)`
Shrinks the `String` pointed to by the parameter so that its `size` matches its `capacity`.

`void fromBuffer(String *, string_value_type const *)`
Copies the string pointed to by the `string_value_type const *` parameter into the `String` pointed to by the first parameter.

`void toBuffer(String const *, string_value_type *, string_size_type)`
Copies the content contained by the `String` pointed to by the first parameter to the buffer pointed to by the second parameter. The buffer pointed to by the second parameter is of the size passed into the third parameter. Note that that size refers to the entire buffer size including the '\0' character at the end (if the buffer even has one of those). The buffer will contain as many characters from the `String` pointed to by the first parameter as the buffer can

hold - 1, as the last byte in the buffer will be ‘\0’ to form a valid C-String.

`void append(String *, string_value_type const *)`
Appends the C-String from the second parameter to the `String` pointed to by the first parameter.

`int compare(String const *, string_value_type const *)`
Compares the `String` pointed to by the first parameter with the C-String from the second parameter. Returns 0 if the strings are equal. Returns a negative value if the first character that does not match has a lower value in the `String` than in the `string_value_type const *`. Returns a positive value if the first character that does not match has a greater value in the `String` than in the `string_value_type const *`.

`bool equals(String const *, string_value_type const *)`
Returns a non-zero value if the `String` pointed to by the first parameter is equal to the C-String from the second parameter. Returns 0 otherwise.

`void fillWith(String *, string_value_type)`
Fills all the bytes in the `String` pointed to by the first parameter with the character from the second parameter until the first ‘\0’ character Is reached.

`void swap(String *, String *)`
Swaps the contents of the 2 `Strings` pointed to by the parameters.

`void pushBack(String *, string_value_type)`
Appends the character from the second parameter to the `String` pointed to by the first parameter.

`string_value_type popBack(String *)`
Removes the last character (the one before the first ‘\0’ character) from the `String` pointed to by the parameter and returns a copy of the removed character.

`void reverse(String *)`
Reverses the `String` pointed to by the parameter.

`void readLine(String *, FILE *)`
Reads one line from the stream passed into the second parameter into the `String` pointed to by the first parameter.