Assignment 1 – Own string Class

NourUtilString

v1.0.0

Nour Ahmed
Matrikal-Nr.: 5200991

implementation of own string class.

This report presents an implementation of a class named string. This class behavior will be similar and compatible to the std::string. Here the prototypes for the class, its methods and eventually any macros, constants, or global variables needed are descried.

Chapter 1

my_cpp_string

Implementation of Own String Class

In this presents an implementation of a class named `util::string`. This class behavior is similar to the `std::string` and both `std::string` and this `util::string` class are compatible.

Full and detailed examples of uses and tests of the class `util::string` are given in the `main.cpp` file. Each method and operator is very carefully tested (e.g., concatenating different strings, ..., etc).

An example test run is shown in the following screenshot:

Note that the **terminal output is colored** (using `ANSI escape codes`) for better visibility.

Doxygen generated documentation (in html and LaTeX formats) can be found at `doc/html/index.html` and `doc/latex/refman.pdf`, respectively. The configuration file `Doxyfile` is used with the Doxygen generation tool.

The following design and implementation criteria are followed:

- **No C/C++ standard functions or classes are used** to realize `util::string` class. This include, e.g., `strcmp`, `strlen` and of course using `std::string` as an internal representation of `util::string`.
  - This means own functions/methods are developed and implemented to calculate the length of a `char*`, to compare character sequences or to copy them full or partially.
- For now, **no error handling** (e.g., accessing an invalid index by using operator []) is implemented. This may be done later.
  - **Use this class at your own risk** :).
- The code follows `LLVM Coding Standards`.
- The *`sanke_case`* naming convention is used for variable and function names (with few exceptions).
- **Use this class at your own risk** :).

1.1 General Functionality

- The class `util::string` is implemented inside the two files `utilstring.cpp` and `utilstring.h`
- Class string is within the namespace `util`
- The memory management is done by using a pointer (`internal_buffer`) pointing to the data type char. `char*` are (normally) null terminated. This means, that the last character is always a `\0` (NULL character) which marks the end of a char sequence. This character is never printed as it just allows for detecting whether the end of a char sequence has been reached. The string is always null-terminate (internally!)
- Initially, the class provides memory for 10 printable characters. Note that this default value is provided by the constant `INITIAL_SIZE` (defined at the top of `utilstring.h`). It can be changed if another value is desired.
- A relatively simple concept is designed and implemented to extend the internal memory if `util::string` has to store more than 10 characters.

1.2 Constructors

The following constructors are implemented:

- `string()` : Default constructor with empty initialization
- `string(size_t intialSize= INITIAL_SIZE)` : constructor with parameter for the initial memory size to initialization with.
- `string(const string&)` : Copy constructor: Creates a deep copy of a passed string.
- `string(const char*)` : Constructor with parameter `const char*`.
- `string(const std::string&)` : Constructor with parameter std::string.

1.3 Operators

The following operators are implemented:

- Operator + and += such that string, std::string and (const) char* can be added
- Assignment operator = such that string, std::string and (const) char* can be assigned
- Comparison operators == and != such that comparisons with `util::string`, `std::string` and `(const) char*` are possible. With respect to the last two cases, `std::string` and `const char*` may both be LHS as well as RHS arguments.
- operator [] to access individual characters of `util::string` object.
- Streaming operator << to print `util::string` to `std::cout`.

1.4 Methods

The following methods are implemented:

- `clear()`: Clears your string object
- `substr(pos, length)`: Returns a substring object of type util::string which starts at pos. Parameter length specifies the amount of characters of the new `util::string` to be returned.
- `length()`: Returns the amount of characters of your string excluding `\0`. Might be smaller than the actual reserved memory.
- `c_str()`: Allows raw access to the internal C-string respectively the `char*` pointer

1.5 References

- Standard Strings library : https://en.cppreference.com/w/cpp/string
- C++ ISO Standard https://isocpp.org/std/the-standard
- C++ documentation - DevDocs : https://devdocs.io/cpp/
- LLVM Coding Standards: https://llvm.org/docs/CodingStandards.html
- sanke_case convention : https://en.wikipedia.org/wiki/Snake_case
- Markdown Basic Syntax : https://www.markdownguide.org/basic-syntax
- Doxygen : https://www.doxygen.nl/index.html

Chapter 2

Namespace Index

2.1 Namespace List

Here is a list of all namespaces with brief descriptions:

Chapter 3

Class Index

3.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 4

File Index

4.1 File List

Here is a list of all files with brief descriptions:

Chapter 5

Namespace Documentation

5.1 util Namespace Reference

Classes

- class string

Functions

- std::ostream & operator<< (std::ostream &iostream, const util::string &myString)
- bool operator== (const std::string &lhsString, const util::string &rhsString)
- bool operator== (const char *lhsCharArray, const util::string &rhsString)
- bool operator!= (const std::string &lhsString, const util::string &rhsString)
- bool operator!= (const char *lhsCharArray, const util::string &rhsString)
- void deepCopy (char *rawCharTarget, const char *rawCharSource, size_t destStartPosition, size_t srcEnd↩ Position)
- void printHeader (const char *text)
- void printSubHeader (const char *text)
- void printTestCase (const char *text)

5.1.1 Function Documentation

5.1.1.1 deepCopy()

```
void util::deepCopy (
            char * rawCharTarget,
            const char * rawCharSource,
            size_t destStartPosition,
            size_t srcEndPosition )
```

fill rawCharTarget with rawCharSource starting from startPosition

Notes: > rawCharTarget contents will be changed > the rawCharTarget is assumed to be big enough to hold the rawCharSource (i.e., its size is larger than or equal to that of the rawCharSource)

destStartPosition default is to first location of the destRawChar srcEndPosition default is to last character (before the \0) of the srcRawChar

Definition at line 386 of file utilstring.cpp.

5.1.1.2 operator"!=() [1/2]

```
bool util::operator!= (
            const char * lhsCharArray,
            const util::string & rhsString )
```

Definition at line 363 of file utilstring.cpp.

5.1.1.3 operator"!=() [2/2]

```
bool util::operator!= (
            const std::string & lhsString,
            const util::string & rhsString )
```

Definition at line 358 of file utilstring.cpp.

5.1.1.4 operator<<()

```
std::ostream & util::operator<< (
            std::ostream & iostream,
            const util::string & myString )
```

Definition at line 342 of file utilstring.cpp.

5.1.1.5 operator==() [1/2]

```
bool util::operator== (
            const char * lhsCharArray,
            const util::string & rhsString )
```

Definition at line 352 of file utilstring.cpp.

5.1.1.6 operator==() [2/2]

```
bool util::operator== (
            const std::string & lhsString,
            const util::string & rhsString )
```

Definition at line 347 of file utilstring.cpp.

5.1.1.7 printHeader()

```
void util::printHeader (
            const char * text )
```

utility functions for printing nice text output

ANSI Escape Sequences are used to color the console text, it works for windows and Linux. For Windows, you need to run the program in the new terminal as the old one does not support these codes. see: https://gist.github.com/fnky/458719343aabd01cfb17a3a4f7296797

Definition at line 419 of file utilstring.cpp.

5.1.1.8 printSubHeader()

```
void util::printSubHeader (
            const char * text )
```

Definition at line 436 of file utilstring.cpp.

5.1.1.9 printTestCase()

```
void util::printTestCase (
            const char * text )
```

Definition at line 442 of file utilstring.cpp.

Chapter 6

Class Documentation

6.1 util::string Class Reference

`#include <utilstring.h>`

Public Member Functions

- string (size_t intialSize=INITIAL_SIZE)
- string (const char *)
- string (const std::string &)
- string (const string &)
- ~string (void)
- void intialize_string (size_t length=0)
- void deepCopy (const char *rawChar, size_t startPosition=0)
- string substr (size_t pos, size_t length)
- char * c_str () const
- void clear ()
- size_t length () const
- size_t size () const
- size_t capacity () const
- string operator+ (const string &rhsString)
- string operator+ (const std::string &rhsString)
- string operator+ (const char *strInstance)
- string & operator+= (const string &rhsString)
- string & operator+= (const std::string &rhsString)
- string & operator+= (const char *strInstance)
- string & operator= (const string &rhsString)
- string & operator= (const char *rhsCharArray)
- string & operator= (const std::string &rhsString)
- bool operator== (const string &rhsString)
- bool operator== (const std::string &rhsString)
- bool operator== (const char *charArray)
- bool operator!= (const string &rhsString)
- bool operator!= (const std::string &rhsString)
- bool operator!= (const char *charArray)
- const char operator[ ] (size_t position)

Static Public Member Functions

- static int compare (const char *s1, const char *s2)
- static size_t rawSize (const char *rawChar)

Friends

- std::ostream & operator<< (std::ostream &iostream, const util::string &myString)
- bool operator== (const std::string &lhsString, const util::string &rhsString)
- bool operator== (const char *lhsCharArray, const util::string &rhsString)
- bool operator!= (const std::string &lhsString, const util::string &rhsString)
- bool operator!= (const char *lhsCharArray, const util::string &rhsString)

6.1.1 Detailed Description

Definition at line 32 of file utilstring.h.

6.1.2 Constructor & Destructor Documentation

6.1.2.1 string() [1/4]

```
util::string::string (
            size_t intialSize = INITIAL_SIZE )
```

default constructor with empty initialization

Default Constructor

Definition at line 28 of file utilstring.cpp.

6.1.2.2 string() [2/4]

```
util::string::string (
            const char * data )
```

Constructor with parameter const char*

Constructor with char*

Definition at line 32 of file utilstring.cpp.

6.1.2.3 string() [3/4]

```
util::string::string (
            const std::string & data )
```

Constructor with parameter std::string

Definition at line 40 of file utilstring.cpp.

6.1.2.4 string() [4/4]

```
util::string::string (
            const string & data )
```

Copy constructor: Creates a deep copy of a passed string

Definition at line 48 of file utilstring.cpp.

6.1.2.5 ~string()

```
util::string::~string (
            void  )
```

Definition at line 57 of file utilstring.cpp.

6.1.3 Member Function Documentation

6.1.3.1 c_str()

```
char * util::string::c_str ( ) const
```

Definition at line 137 of file utilstring.cpp.

6.1.3.2 capacity()

```
size_t util::string::capacity ( ) const
```

Returns the size of the storage space currently allocated for the string, expressed in terms of bytes.

Definition at line 80 of file utilstring.cpp.

6.1.3.3 clear()

```
void util::string::clear ( )
```

Clears your string object Erases the contents of the string, which becomes an empty string(with a length of 0 characters).

Clears your string object Erases the contents of the string, which becomes an empty string (with a length of 0 characters).

Definition at line 90 of file utilstring.cpp.

6.1.3.4 compare()

```
int util::string::compare (
            const char * lhsCharArray,
            const char * rhsCharArray )  [static]
```

Compares two char * strings lexicographically This function is my own implementation of the std::strcmp() function. Note this function performs a binary comparison of the ASCII code of the characters.

Parameters

str1	primitive C string to be compared.
str2	primitive C string to be compared with.

Returns

an integral value indicating the relationship between the strings: <0 : the first character that does not match has a lower value in ptr1 than in ptr2 0 : the contents of both strings are equal >0 : the first character that does not match has a greater value in ptr1 than in ptr2

Definition at line 169 of file utilstring.cpp.

6.1.3.5 deepCopy()

```
void util::string::deepCopy (
            const char * rawChar,
            size_t startPosition = 0 )
```

Design and implementation of a concept to extend the internal memory if util::string has to store more than the default INITIAL_SIZE characters note that the function copy the passed char array starting from the startPosition (i.e. it can write starting from any position in the internal string buffer) startPosition default is 0

Definition at line 114 of file utilstring.cpp.

6.1.3.6 intialize_string()

```
void util::string::intialize_string (
            size_t length = 0 )
```

Definition at line 63 of file utilstring.cpp.

6.1.3.7 length()

```
size_t util::string::length ( ) const
```

Returns the amount of characters of your string excluding \0. Might be smaller than the actual reserved memory.

Definition at line 73 of file utilstring.cpp.

6.1.3.8 operator"!=() [1/3]

```
bool util::string::operator!= (
            const char * charArray )
```

Definition at line 240 of file utilstring.cpp.

6.1.3.9 operator"!=() [2/3]

```
bool util::string::operator!= (
            const std::string & rhsString )
```

Definition at line 233 of file utilstring.cpp.

6.1.3.10 operator"!=() [3/3]

```
bool util::string::operator!= (
            const string & rhsString )
```

Definition at line 227 of file utilstring.cpp.

6.1.3.11 operator+() [1/3]

```
string util::string::operator+ (
            const char * strInstance )
```

Definition at line 328 of file utilstring.cpp.

6.1.3.12 operator+() [2/3]

```
string util::string::operator+ (
            const std::string & rhsString )
```

Definition at line 318 of file utilstring.cpp.

6.1.3.13 operator+() [3/3]

```
string util::string::operator+ (
            const string & rhsString )
```

Operator + such that string, std::string and (const) char* can be added

Definition at line 308 of file utilstring.cpp.

6.1.3.14 operator+=() [1/3]

```
string & util::string::operator+= (
            const char * rhsString )
```

concatenating util::string and const char*

Definition at line 292 of file utilstring.cpp.

6.1.3.15 operator+=() [2/3]

```
string & util::string::operator+= (
            const std::string & rhsString )
```

Definition at line 273 of file utilstring.cpp.

6.1.3.16 operator+=() [3/3]

```
string & util::string::operator+= (
            const string & rhsString )
```

Definition at line 255 of file utilstring.cpp.

6.1.3.17 operator=() [1/3]

```
string & util::string::operator= (
            const char * rhsCharArray )
```

Definition at line 195 of file utilstring.cpp.

6.1.3.18 operator=() [2/3]

```
string & util::string::operator= (
            const std::string & rhsString )
```

Definition at line 200 of file utilstring.cpp.

6.1.3.19 operator=() [3/3]

```
string & util::string::operator= (
            const string & rhsString )
```

Definition at line 190 of file utilstring.cpp.

6.1.3.20 operator==() [1/3]

```
bool util::string::operator== (
            const char * charArray )
```

Definition at line 219 of file utilstring.cpp.

6.1.3.21 operator==() [2/3]

```
bool util::string::operator== (
            const std::string & rhsString )
```

Definition at line 212 of file utilstring.cpp.

6.1.3.22 operator==() [3/3]

```
bool util::string::operator== (
            const string & rhsString )
```

Definition at line 206 of file utilstring.cpp.

6.1.3.23 operator[]()

```
const char util::string::operator[] (
            size_t position )
```

Definition at line 248 of file utilstring.cpp.

6.1.3.24 rawSize()

```
size_t util::string::rawSize (
            const char * rawChar )  [static]
```

Get the amount of characters of a raw char* string excluding the terminating \0.

Definition at line 97 of file utilstring.cpp.

6.1.3.25 size()

```
size_t util::string::size ( ) const
```

Get the length of the string synonyme to length()

Definition at line 72 of file utilstring.cpp.

6.1.3.26 substr()

```
string util::string::substr (
            size_t pos,
            size_t length )
```

Definition at line 146 of file utilstring.cpp.

6.1.4 Friends And Related Function Documentation

6.1.4.1 operator"!= [1/2]

```
bool operator!= (
            const char * lhsCharArray,
            const util::string & rhsString )  [friend]
```

Definition at line 363 of file utilstring.cpp.

6.1.4.2 operator"!= [2/2]

```
bool operator!= (
            const std::string & lhsString,
            const util::string & rhsString )  [friend]
```

Definition at line 358 of file utilstring.cpp.

6.1.4.3 operator<<

```
std::ostream & operator<< (
            std::ostream & iostream,
            const util::string & myString )  [friend]
```

Definition at line 342 of file utilstring.cpp.

6.1.4.4 operator== [1/2]

```
bool operator== (
            const char * lhsCharArray,
            const util::string & rhsString )  [friend]
```

Definition at line 352 of file utilstring.cpp.

6.1.4.5 operator== [2/2]

```
bool operator== (
            const std::string & lhsString,
            const util::string & rhsString )  [friend]
```

Definition at line 347 of file utilstring.cpp.

The documentation for this class was generated from the following files:

- utilstring.h
- utilstring.cpp

Chapter 7

File Documentation

7.1 main.cpp File Reference

: test of own implementation of string class

```
#include "utilstring.h"
#include <iostream>
```

Functions

- int main ()

7.1.1 Detailed Description

: test of own implementation of string class

==========

Author

: Nour Ahmed @email : nahmed@stud.hs-bremen.de, nour @repo : https://github.com/nouremara/cpp_mystring @repo : @createdOn : 23.11.2022

Version

: 1.0.0 @description :

Defines the entry point for the NourUtilString application In this application the class util::string is used and tested. Each method and operator is tested with all possible uasges (e.g., concatenating different strings etc.) Note: For this task no error handling is required.Example: Accessing

7.1.2 an invalid index by using operator [ ]

Definition in file main.cpp.

7.1.3 Function Documentation

7.1.3.1 main()

```
int main ( )
```

Definition at line 28 of file main.cpp.

7.2 main.cpp

Go to the documentation of this file.

```
00001 // Nour Ahmed
00002 // Matrikal-Nr.: 5200991
00003 // Assignment 1 - Own string Class
00004
00024 #include "utilstring.h"
00025
00026 #include <iostream>
00027
00028 int main() {
00029     char charArray[] = "text in a const char array";
00030     std::string stdString("another text in a std::string");
00031
00032     // instantiate objects
00033     util::string string1;
00034     util::string string2("initializing with const char array");
00035     util::string string3(charArray);
00036     util::string string4(stdString);
00037     util::string string5(string4);
00038
00039     util::printHeader("NourUtilString Application");
00040     std::cout « "\033[1;30;106m- Nour Ahmed
            -" « std::endl;
00041     std::cout « "- Matrikal-Nr.: 5200991                                                    -" «
      std::endl;
00042     std::cout « "- Assignment 1 - Own string Class                                          -" «
      std::endl;
00043     std::cout «
      "-------------------------------------------------------------------------------\033[0m\n\n";
00044
00045     // Test Object Instantiation ------------------------------------------
00046     util::printSubHeader("Variable used for testing and their values");
00047     std::cout « "Variable used for testing and their values" « std::endl;
00048
00049     util::printTestCase("charArray");
00050     std::cout « "charArray (size: " « util::string::rawSize(charArray) « ")  : " « charArray «
      std::endl;
00051
00052     util::printTestCase("stdString");
00053     std::cout « "charArray (size: " « stdString.length() « ")  : " « stdString « std::endl;
00054
00055     std::cout « "-------------------------------------------------------------------------------\n\n";
00056
00057     util::printSubHeader("Test object constructors and initialization");
00058
00059     util::printTestCase("default constructor");
00060     std::cout « "\tstring1 (size: " « string1.size() « ")  : " « string1 « std::endl;
00061
00062     util::printTestCase("constructor with const char*");
00063     std::cout « "string2 (size: " « string2.size() « ") : " « string2 « std::endl;
00064
00065     util::printTestCase("constructor with std::string");
00066           std::cout « "string3 (size: " « string3.size() « ") : " « string3 « std::endl;
00067
00068     util::printTestCase("constructor with char array");
00069     std::cout « "string4 (size: " « string4.size() « ") : " « string4 « std::endl;
00070
00071     util::printTestCase("constructor with util::string");
00072     std::cout « "string5 (size: " « string5.size() « ") : " « string4 « std::endl;
00073     std::cout « "-------------------------------------------------------------------------------\n\n";
00074
00075
```

```
00076     // Test member methods ------------------------------------------------------
00077     util::printSubHeader("Test Member Methods");
00078 
00079 
00080     util::printTestCase("length()");
00081     std::cout « "string2 (size: " « string2.size() « ", capacity: " « string2.capacity() « ") : " «
      string2 « std::endl;
00082 
00083     util::printTestCase("size()");
00084     std::cout « "string2 (size: " « string2.size() « ", capacity: " « string2.capacity() « ") : " «
      string2 « std::endl;
00085 
00086     util::printTestCase("capacity()");
00087     std::cout « "string2 (size: " « string2.size() « ", capacity: " « string2.capacity() « ") : " «
      string2 « std::endl;
00088 
00089     util::string temp = string2.substr(3, 5);
00090     util::printTestCase("substr()");
00091     std::cout « "string2.substr(3,5) \t -> " « temp « std::endl;
00092 
00093     util::printTestCase("c_str()");
00094     std::cout « "string2.c_str() \t -> " « string2.c_str() « std::endl;
00095 
00096     string2.clear();
00097     util::printTestCase("clear()");
00098     std::cout « "string2.clear() -> string2 (size: " « string2.size() « ", capacity: " «
      string2.capacity() « ") : content: " « string2 « std::endl;
00099     std::cout « "-------------------------------------------------------------------------------\n\n";
00100 
00101     // Test operators -----------------------------------------------------
00102     util::printSubHeader("Test operators");
00103 
00104     util::printTestCase("operator «");
00105     std::cout « "std::cout « util::string « int « std::string « char *:\n"
00106         « "    string3 (size: " « string3.size() « ", capacity: " « string3.capacity() « ") : content:
      " « string3
00107         « stdString « ", "
00108         « charArray
00109         « std::endl;
00110     std::cout « "-------------------------------------------------------------------------------\n\n";
00111 
00112     util::printTestCase("operator +");
00113     std::cout « "\n\tutil::string + util::string \t: string2 + string3 -> " « string2 + string3 «
      std::endl;
00114 
00115     string5 = string5 + " see how + operator with char * works";
00116     std::cout « "\tutil::string + const char* \t: string5 = string5 + const char* -> (size: "  «
      string5.size() « ") : " « string5;
00117     std::cout «
      "\n-------------------------------------------------------------------------------\n\n";
00118 
00119     string4 += string3;
00120     util::printTestCase("operator +=");
00121     std::cout « "\n\twith util::string\t: string4 += string3 -> (size: " « string4.size() « ") : " «
      string4 « std::endl;
00122 
00123     string4 += " here += operator is used to add more text in char *";
00124     std::cout « "\twith const char* \t: string4 += const char* -> (size: " « string4.size() « ") : " «
      string4 « std::endl;
00125     std::cout « "-------------------------------------------------------------------------------\n\n";
00126 
00127 
00128     string1 = string4;
00129     string2 = "more text for testing";
00130     string3 = std::string("text for std::string assignment");
00131 
00132     util::printTestCase("operator =");
00133     std::cout « "\n\tutil::string = util::string\t string1 = string4 -> string1 (size: " «
      string1.size() « "): " « string1 « std::endl;
00134     std::cout « "\tutil::string = const char* \t string2 = \"...\" -> string2 (size: " «
      string2.size() « "): " « string2 « std::endl;
00135     std::cout « "\tutil::string = std::string \t string3 = std::string(\"...\") -> string3 (size: " «
      string3.size() « "): " « string3 « std::endl;
00136     std::cout « "-------------------------------------------------------------------------------\n\n";
00137 
00138     string1 = string2;
00139     util::printTestCase("operator ==");
00140     std::cout « "\n\tutil::string == util::string \t string1   == string2   -> " « ((string1 ==
      string2) ? "true" : "false") « std::endl;
00141     std::cout « "\tutil::string == std::string  \t string1   == stdString -> " « ((string1 ==
      stdString) ? "true" : "false") « std::endl;
00142     std::cout « "\tstd::string  == util::string \t stdString == string1   -> " « ((stdString ==
      string1) ? "true" : "false") « std::endl;
00143     std::cout « "\tutil::string == const char*  \t string1   == charArray -> " « ((string1 ==
      charArray) ? "true" : "false") « std::endl;
00144     std::cout « "\tconst char*  == util::string \t charArray == string1   -> " « ((charArray ==
      string1) ? "true" : "false") « std::endl;
```

```
00145     std::cout « "-------------------------------------------------------------------------------\n\n";
00146
00147     util::printTestCase("operator !=");
00148     std::cout « "\n\tutil::string != util::string \t string1   != string2   -> " « ((string1 !=
      string2) ? "true" : "false") « std::endl;
00149     std::cout « "\tutil::string != std::string  \t string1   != stdString -> " « ((string1 !=
      stdString) ? "true" : "false") « std::endl;
00150     std::cout « "\tstd::string  != util::string \t stdString != string1   -> " « ((stdString !=
      string1) ? "true" : "false") « std::endl;
00151     std::cout « "\tutil::string != const char*  \t string1   != charArray -> " « ((string1 !=
      charArray) ? "true" : "false") « std::endl;
00152     std::cout « "\tconst char*  != util::string \t charArray != string1   -> " « ((charArray !=
      string1) ? "true" : "false") « std::endl;
00153     std::cout « "-------------------------------------------------------------------------------\n\n";
00154
00155     util::printTestCase("operator []");
00156     std::cout « "\n\tstring1: " « string1 « "-> string1[0]: " « string1[0] « std::endl;
00157     std::cout « "\tstring2: " « string2 « "-> string2[3]: " « string2[3] « std::endl;
00158     std::cout « "\tstring3: " « string3 « "-> string3[50]: " « string3[50] « std::endl;
00159     std::cout « "-------------------------------------------------------------------------------\n\n";
00160
00161     // Test utility functions
00162     util::printSubHeader("Test utility functions");
00163
00164     char s1[100] = "programming ", s2[] = "is awesome";
00165     std::cout « "\ts1 (size: " « util::string::rawSize(s1) « ", capacity: 100) : content: " « s1 «
      std::endl;
00166     std::cout « "\ts2 (size: " « util::string::rawSize(s2) « ", capacity: "«
      util::string::rawSize(s2)+1 «") : content: " « s2 « std::endl;
00167
00168     util::printTestCase("util::deepCopy()");
00169     util::deepCopy(s1, s2);
00170     std::cout « "\n\tdeepCopy(s1, s2) -> s1 (size: " « util::string::rawSize(s1) « ", capacity: 100) :
      content: " « s1 « std::endl;
00171
00172     util::printTestCase("util::rawSize()");
00173     std::cout « "\n\tutil::string::rawSize(s1) -> " « util::string::rawSize(s1) « std::endl;
00174
00175     util::printTestCase("util::string::compare()");
00176     std::cout « "\n\tutil::string::compare(s1,s2) -> " « util::string::compare(s1, s2) « std::endl;
00177     std::cout « "\tutil::string::compare(s1,s1) -> " « util::string::compare(s1, s1) « std::endl;
00178     std::cout « "-------------------------------------------------------------------------------\n\n";
00179
00180     util::printSubHeader("Test utility functions");
00181     util::printTestCase("util::printHeader()");      std::cout « std::endl;
00182     util::printTestCase("util::printSubHeader()");  std::cout « std::endl;
00183     util::printTestCase("util::printTestCase()");   std::cout « std::endl;
00184     std::cout « "\tThese functions are used to print the above colored headers :)" « std::endl;
00185     std::cout « "-------------------------------------------------------------------------------\n\n";
00186
00187     return 0;
00188 }
```

7.3 README.md File Reference

7.4 utilstring.cpp File Reference

: implementation of own string class.

```
#include "utilstring.h"
```

Namespaces

- namespace util

Functions

- std::ostream & util::operator<< (std::ostream &iostream, const util::string &myString)
- bool util::operator== (const std::string &lhsString, const util::string &rhsString)
- bool util::operator== (const char *lhsCharArray, const util::string &rhsString)
- bool util::operator!= (const std::string &lhsString, const util::string &rhsString)
- bool util::operator!= (const char *lhsCharArray, const util::string &rhsString)
- void util::deepCopy (char *rawCharTarget, const char *rawCharSource, size_t destStartPosition, size_t srcEndPosition)
- void util::printHeader (const char *text)
- void util::printSubHeader (const char *text)
- void util::printTestCase (const char *text)

7.4.1 Detailed Description

: implementation of own string class.

==========

Author

: Nour Ahmed @email : nahmed@stud.hs-bremen.de, nourbrm02@gmail.com @repo : https://github.com/nouremara/cpp_mystring @createdOn : 23.11.2022

Version

: 1.0.0 @description : implementation of own string class

This file presents an implementation of a class named string. This class behavior will be similar to the std::string and both

7.4.2 std::string and this string class are compatible.

Definition in file utilstring.cpp.

7.5 utilstring.cpp

Go to the documentation of this file.

```
00001 // Nour Ahmed
00002 // Matrikal-Nr.: 5200991
00003 // Assignment 1 - Own String Class
00004
00020 #include "utilstring.h"
00021
00022 namespace util {
00023 /*==========================================================================*
00024  *                          Constructors                                    *
00025  *==========================================================================*/
00026
00028 string::string(size_t intialSize) { intialize_string(intialSize); }
00029 //---------------------------------------------------------------
00030
00032 string::string(const char *data) {
00033   intialize_string(
00034       rawSize(data)); // ensure string is initialized before using it
00035   deepCopy(data);     // copy passed array to the string
```

```
00036 }
00037 //-------------------------------------------------------------
00038
00040 string::string(const std::string &data) {
00041   intialize_string(
00042       rawSize(data.c_str())); // ensure string is initialized before using it
00043   deepCopy(data.c_str());     // copy passed array to the string
00044 }
00045 //-------------------------------------------------------------
00046
00048 string::string(const string &data) {
00049   intialize_string(
00050       rawSize(data.c_str())); // ensure string is initialized before using it
00051   deepCopy(data.c_str());     // copy passed array to the string
00052 }
00053
00054 /*==========================================================================*
00055  *                          Destructor                                      *
00056  *==========================================================================*/
00057 string::~string(void) { delete[] internal_buffer; }
00058
00059 /*==========================================================================*
00060  *                          Methods                                         *
00061  *==========================================================================*/
00062
00063 void string::intialize_string(size_t length) {
00064   internal_buffer = new char[length + 1];
00065   buffer_size = length + 1;
00066
00067   // initialize an empty string
00068   internal_buffer[0] = '\0';
00069 }
00070 //-------------------------------------------------------------
00071
00072 size_t string::size() const { return rawSize(internal_buffer); }
00073 size_t string::length() const { return rawSize(internal_buffer); }
00074 //-------------------------------------------------------------
00075
00080 size_t string::capacity() const {
00081     return buffer_size;
00082 }
00083 //-------------------------------------------------------------
00084
00090 void string::clear(){
00091     // we only need to set the termination character to the first postion
00092     // to indicate that the string is empty
00093     // initialize an empty string
00094     internal_buffer[0] = '\0';
00095 }
00096
00097 size_t string::rawSize(const char *rawChar) {
00098   size_t length = 0;
00099   while (rawChar[length] != '\0') {
00100     length++;
00101   }
00102
00103   return length;
00104 }
00105 //-------------------------------------------------------------
00106
00114 void string::deepCopy(const char *rawChar, size_t startPosition) {
00115   // check if internalData is of enough size to accommodate the passed array
00116   size_t rawCharSize = rawSize(rawChar);
00117   if (rawCharSize > size()) { // more space is needed
00118     // delete current internalData
00119     delete[] internal_buffer;
00120
00121     // re-initialize the string with the required size
00122     intialize_string(rawCharSize);
00123   }
00124
00125   // copy the passed array to the newly allocated internalData
00126   int j = startPosition;
00127   while (rawChar[j] != '\0') {
00128     internal_buffer[j] = rawChar[j];
00129     j++;
00130   }
00131
00132   internal_buffer[j] = '\0'; // ensure destination string is null terminated
00133   // string_size = rawCharSize;     // set string size to the new one
00134 }
00135 //-------------------------------------------------------------
00136
00137 char *string::c_str() const { return internal_buffer; }
00138 //-------------------------------------------------------------
00139
00140 // Returns a pointer to an array that contains a null-terminated
```

```
00141 // sequence of characters(i.e., a C-string) representing the current
00142 // value of the string object.
00143 // Returns a substring object of type util::string which
00144 // starts at pos. Parameter length specifies the amount of
00145 // characters of the new util::string to be returned.
00146 string string::substr(size_t pos, size_t length) {
00147   string substring(length);
00148
00149   util::deepCopy(substring.c_str(), internal_buffer + pos, 0, length);
00150
00151   return substring;
00152 }
00153
00154 //-------------------------------------------------------------
00155
00169 int string::compare(const char *lhsCharArray, const char *rhsCharArray) {
00170   // convert both pointers from `char*` to `unsigned char*`
00171   // needed for the difference calculations
00172   const unsigned char *p1 = (const unsigned char *)lhsCharArray;
00173   const unsigned char *p2 = (const unsigned char *)rhsCharArray;
00174
00175   // check if characters differ, or end of the first string (a terminating null)
00176   // is reached
00177   while (*p1 && *p1 == *p2) {
00178     // proceed to the next pair of characters
00179     ++p1, ++p2;
00180   }
00181
00182   // return the ASCII difference
00183   return (*p1 > *p2) - (*p2 > *p1);
00184 }
00185
00186 /*=============================================================================*
00187  *                           Operators                                         *
00188  *=============================================================================*/
00189
00190 string &string::operator=(const string &rhsString) {
00191   deepCopy(rhsString.c_str());
00192   return *this;
00193 }
00194
00195 string &string::operator=(const char *rhsCharArray) {
00196   deepCopy(rhsCharArray);
00197   return *this;
00198 }
00199
00200 string &string::operator=(const std::string &rhsString) {
00201   deepCopy(rhsString.c_str());
00202   return *this;
00203 }
00204 //-------------------------------------------------------------
00205
00206 bool string::operator==(const string &rhsString) {
00207   // if (string_size != rhsString.size()) return false;
00208   // note that compare returns 0 when the two strings are equal
00209   return !compare(internal_buffer, rhsString.c_str());
00210 }
00211
00212 bool string::operator==(const std::string &rhsString) {
00213   // if (string_size != rhsString.size()) return false;
00214
00215   // note that compare returns 0 when the two strings are equal
00216   return !compare(internal_buffer, rhsString.c_str());
00217 }
00218
00219 bool string::operator==(const char *charArray) {
00220   // if (string_size != rhsString.size()) return false;
00221
00222   // note that compare returns 0 when the two strings are equal
00223   return !compare(internal_buffer, charArray);
00224 }
00225 //-------------------------------------------------------------
00226
00227 bool string::operator!=(const string &rhsString) {
00228   // if (string_size != rhsString.size()) return false;
00229   // note that compare returns 0 when the two strings are equal
00230   return compare(internal_buffer, rhsString.c_str());
00231 }
00232
00233 bool string::operator!=(const std::string &rhsString) {
00234   // if (string_size != rhsString.size()) return false;
00235
00236   // note that compare returns 0 when the two strings are equal
00237   return compare(internal_buffer, rhsString.c_str());
00238 }
00239
00240 bool string::operator!=(const char *charArray) {
```

```
00241   // if (string_size != rhsString.size()) return false;
00242
00243   // note that compare returns 0 when the two strings are equal
00244   return compare(internal_buffer, charArray);
00245 }
00246 //-------------------------------------------------------------
00247
00248 const char string::operator[](size_t position) {
00249   if (position > size())
00250     return '\0';
00251   return internal_buffer[position];
00252 }
00253 //-------------------------------------------------------------
00254
00255 string &string::operator+=(const string &rhsString) {
00256   size_t total_size = size() + rhsString.size() + 1;
00257   char *temp = new char[total_size];
00258
00259   util::deepCopy(temp, internal_buffer, 0);
00260   util::deepCopy(temp, rhsString.c_str(), size());
00261
00262   delete[] internal_buffer;
00263
00264   internal_buffer = temp;
00265   buffer_size = total_size;
00266   //std::cout « "\n\nbuffer_size " « buffer_size « std::endl;
00267   //std::cout « "(size: " « size() « ") : " « internal_buffer « std::endl;
00268
00269   return *this;
00270 }
00271 //-------------------------------------------------------------
00272
00273 string& string::operator+=(const std::string& rhsString) {
00274     size_t total_size = size() + rhsString.size() + 1;
00275     char* temp = new char[total_size];
00276
00277     util::deepCopy(temp, internal_buffer, 0);
00278     util::deepCopy(temp, rhsString.c_str(), size());
00279
00280     delete[] internal_buffer;
00281
00282     internal_buffer = temp;
00283     buffer_size = total_size;
00284
00285     return *this;
00286 }
00287 //-------------------------------------------------------------
00288
00292 string &string::operator+=(const char *rhsString) {
00293     size_t total_size = size() + rawSize(rhsString) + 1;
00294     char* temp = new char[total_size];
00295
00296     util::deepCopy(temp, internal_buffer, 0);
00297     util::deepCopy(temp, rhsString, size());
00298
00299     delete[] internal_buffer;
00300
00301     internal_buffer = temp;
00302     buffer_size = total_size;
00303
00304     return *this;
00305 }
00306 //-------------------------------------------------------------
00307
00308 string string::operator+(const string& rhsString) {
00309     string temp(size() + rhsString.size());
00310
00311     util::deepCopy(temp.c_str(), internal_buffer, 0);
00312     util::deepCopy(temp.c_str(), rhsString.c_str(), size());
00313
00314     return temp;
00315 }
00316 //-------------------------------------------------------------
00317
00318 string string::operator+(const std::string& rhsString) {
00319     string temp(size() + rhsString.size());
00320
00321     util::deepCopy(temp.c_str(), internal_buffer, 0);
00322     util::deepCopy(temp.c_str(), rhsString.c_str(), size());
00323
00324     return temp;
00325 }
00326 //-------------------------------------------------------------
00327
00328 string string::operator+(const char* rhsString) {
00329     string temp(size() + rawSize(rhsString));
00330
```

```
00331     util::deepCopy(temp.c_str(), internal_buffer, 0);
00332     util::deepCopy(temp.c_str(), rhsString, size());
00333
00334     return temp;
00335 }
00336
00337 /*=============================================================================*
00338  * non-member (friend) functions and operator methods for the cases            *
00339  * util::string is on the RHS                                                  *
00340  *=============================================================================*/
00341
00342 std::ostream &operator«(std::ostream &iostream, const util::string &myString) {
00343   return (iostream « myString.c_str());
00344 }
00345 //-------------------------------------------------------------
00346
00347 bool operator==(const std::string &lhsString, const util::string &rhsString) {
00348   // note that compare returns 0 when the two strings are equal
00349   return !util::string::compare(lhsString.c_str(), rhsString.c_str());
00350 }
00351
00352 bool operator==(const char *lhsCharArray, const util::string &rhsString) {
00353   // note that compare returns 0 when the two strings are equal
00354   return !util::string::compare(lhsCharArray, rhsString.c_str());
00355 }
00356 //-------------------------------------------------------------
00357
00358 bool operator!=(const std::string &lhsString, const util::string &rhsString) {
00359   // note that compare returns 0 when the two strings are equal
00360   return util::string::compare(lhsString.c_str(), rhsString.c_str());
00361 }
00362
00363 bool operator!=(const char *lhsCharArray, const util::string &rhsString) {
00364   // note that compare returns 0 when the two strings are equal
00365   return util::string::compare(lhsCharArray, rhsString.c_str());
00366 }
00367
00368
00369 /*=============================================================================*
00370 *             Some Utility functions                                          *
00371 *=============================================================================*/
00372
00386 void deepCopy(char *rawCharTarget, const char *rawCharSource,
00387               size_t destStartPosition, size_t srcEndPosition) {
00388   // check and adjust for default values
00389   destStartPosition = (destStartPosition == -1)
00390                           ? util::string::rawSize(rawCharTarget)
00391                           : destStartPosition;
00392   srcEndPosition = (srcEndPosition == -1) ? util::string::rawSize(rawCharSource)
00393                                           : srcEndPosition;
00394
00395   // deep copy rawCharSource into rawCharTarget beginning at startPosition
00396   //        for (size_t j = 0; rawCharSource[j] != '\0'; ++j,
00397   //++destStartPosition) {
00398   for (size_t j = 0; j < srcEndPosition; ++j, ++destStartPosition) {
00399     rawCharTarget[destStartPosition] = rawCharSource[j];
00400   }
00401
00402   // ensure destination string is null terminated
00403   rawCharTarget[destStartPosition] = '\0';
00404 }
00405
00406 /*=============================================================================*
00407  *      Some Utility functions for printing nice text output                   *
00408  *=============================================================================*/
00409
00419 void printHeader(const char *text) {
00420   size_t spaces_needed = (80 - util::string::rawSize(text)) / 2 - 2;
00421
00422   std::cout « "\033[1;30;106m"; // set text and background colors
00423   std::cout « "--------------------------------------------------------------------------------\n-";
00424   for (int i = 0; i < spaces_needed; ++i) {
00425     std::cout « " ";
00426   }
00427   std::cout « text;
00428   for (int i = 0; i < spaces_needed; ++i) {
00429     std::cout « " ";
00430   }
00431   std::cout « " -\n";
00432   std::cout « "--------------------------------------------------------------------------------\n";
00433   std::cout « "\033[0m"; // reset text and background colors
00434 }
00435
00436 void printSubHeader(const char *text) {
00437   std::cout « "\033[32m"; // set text and background colors
00438   std::cout « text;
00439   std::cout « "\033[0m\n"; // reset text and background colors
```

```
00440 }
00441
00442 void printTestCase(const char* text) {
00443     std::cout « "\033[93m  > ["; // set text and background colors
00444     std::cout « text;
00445     std::cout « "]\033[0m \t"; // reset text and background colors
00446 }
00447
00448
00449 } // namespace util
```

7.6 utilstring.h File Reference

: implementation of own string class.

```
#include <cstddef>
#include <iostream>
#include <string>
```

Classes

- class util::string

Namespaces

- namespace util

Macros

- #define INITIAL_SIZE 10

Functions

- void util::deepCopy (char *rawCharTarget, const char *rawCharSource, size_t destStartPosition, size_t src↩ EndPosition)
- void util::printHeader (const char *text)
- void util::printSubHeader (const char *text)
- void util::printTestCase (const char *text)

7.6.1 Detailed Description

: implementation of own string class.

==========

Author

: Nour Ahmed @email : nahmed@stud.hs-bremen.de, nour @repo : https://github.↩ com/nouremara/cpp_mystring @createdOn : 23.11.2022

Version

: 1.0.0 @description :

This file presents an implementation of a class named string. This class behavior will be similar and compatible to the std::string. This file contains the prototypes for the class, its methods and eventually

7.6.2 any macros, constants, or global variables you will need to use it.

Definition in file utilstring.h.

7.6.3 Macro Definition Documentation

7.6.3.1 INITIAL_SIZE

```
#define INITIAL_SIZE 10
```

Definition at line 30 of file utilstring.h.

7.7 utilstring.h

Go to the documentation of this file.

```
00001 // Nour Ahmed
00002 // Matrikal-Nr.: 5200991
00003 // Assignment 1 - Own string Class
00004
00021 #ifndef UTILSTRING_H
00022 #define UTILSTRING_H
00023
00024 #include <cstddef>
00025 #include <iostream>
00026 #include <string>
00027
00028 namespace util {
00029 // Initially, the class shall provide memory for 10 printable characters
00030 #define INITIAL_SIZE 10
00031
00032     class string {
00033         char* internal_buffer;
00034         size_t buffer_size;
00035         // size_t string_size;
00036
00037     public:
00038         /*================== Constructors ===============*/
00039         string(size_t intialSize = INITIAL_SIZE);
00040         string(const char*);
00041         string(const std::string&);
00042         string(const string&);
00044         /*================== Destructor =================*/
00045         ~string(void);
00046
00047         /*================== Methods ====================*/
00048
00049         void intialize_string(size_t length = 0);
00050         void deepCopy(const char* rawChar, size_t startPosition = 0);
00051
00052         // Returns a pointer to an array that contains a null-terminated
00053         // sequence of characters(i.e., a C-string) representing the current
00054         // value of the string object.
00055         // Returns a substring object of type util::string which
00056         // starts at pos. Parameter length specifies the amount of
00057         // characters of the new util::string to be returned.
00058         string substr(size_t pos, size_t length);
00059
00060         // Allows raw access to the internal C-string (through its char* pointer)
00061         char* c_str() const;
00062
00063         //bool compare(const char* charArray) const;
00064         static int compare(const char* s1, const char* s2);
00065
00071         void clear();
00072
00073
00077         static size_t rawSize(const char* rawChar);
```

```
00078
00083         size_t length() const;
00084         size_t size() const;
00090         size_t capacity() const;
00091
00092
00093         /*================== Operators ====================*/
00094
00098         string operator+(const string& rhsString);
00099         string operator+(const std::string& rhsString);
00100         string operator+(const char* strInstance);
00101
00102         string& operator+=(const string& rhsString);
00103         string& operator+=(const std::string& rhsString);
00104         string& operator+=(const char* strInstance);
00105
00106         string& operator=(const string& rhsString);
00107         string& operator=(const char* rhsCharArray);
00108         string& operator=(const std::string& rhsString);
00109
00110         bool operator==(const string& rhsString);
00111         bool operator==(const std::string& rhsString);
00112         bool operator==(const char* charArray);
00113
00114         bool operator!=(const string& rhsString);
00115         bool operator!=(const std::string& rhsString);
00116         bool operator!=(const char* charArray);
00117
00118         const char operator[](size_t position);
00119
00120         /*============================================================================*
00121          *             Non-member function overloads                                  *
00122          *============================================================================*/
00123          // Free operator methods for the cases util::string is on the rhs
00124          // Friendship enables access to private members
00125         friend std::ostream& operator<<(std::ostream& iostream, const util::string& myString);
00126
00127         friend bool operator==(const std::string& lhsString, const util::string& rhsString);
00128         friend bool operator==(const char* lhsCharArray, const util::string& rhsString);
00129
00130         friend bool operator!=(const std::string& lhsString, const util::string& rhsString);
00131         friend bool operator!=(const char* lhsCharArray, const util::string& rhsString);
00132     };
00133
00134     /*============================================================================*
00135      *             Some Utility functions                                         *
00136      *============================================================================*/
00137     //void concat(char* rawCharTarget, char* rawCharSource, size_t startPosition = -1);
00138
00139     void deepCopy(char* rawCharTarget, const char* rawCharSource,
00140         size_t destStartPosition = -1, size_t srcEndPosition = -1);
00141
00142     //-----------------------------------------------
00143     void printHeader(const char* text);
00144     void printSubHeader(const char* text);
00145     void printTestCase(const char* text);
00146
00147
00148 } // namespace util
00149
00150 #endif /* UTILSTRING_H */
```