Lab

Generated by Doxygen 1.9.3

Chapter 1

Hierarchical Index

1.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 2

Class Index

2.1 Class List

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

Chapter 3

File Index

3.1 File List

Here is a list of all files with brief descriptions:

Chapter 4

Class Documentation

4.1 SimpleString Class Reference

`#include <simpleString.h>`

4.1.1 Detailed Description

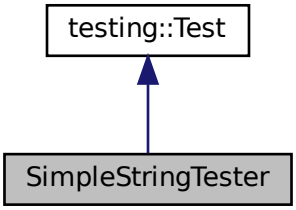
Definition at line 79 of file simpleString.h.

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

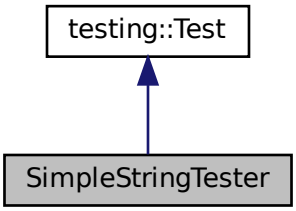
- simpleString.h

4.2 SimpleStringTester Struct Reference

Inheritance diagram for SimpleStringTester:



Collaboration diagram for SimpleStringTester:



4.2.1 Detailed Description

Definition at line 37 of file simpleStringTests.cpp.

The documentation for this struct was generated from the following file:

- simpleStringTests.cpp

Chapter 5

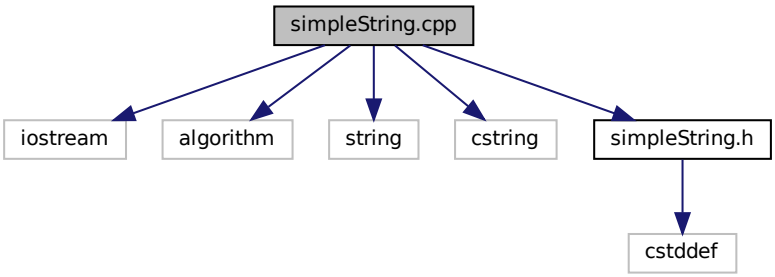
File Documentation

5.1 CMakeLists.txt File Reference

5.2 simpleString.cpp File Reference

```
#include <iostream>
#include <algorithm>
#include <string>
#include <cstring>
#include "simpleString.h"
```

Include dependency graph for simpleString.cpp:



5.3 simpleString.cpp

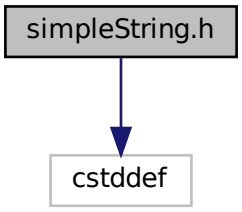
Go to the documentation of this file.

```
00001 #include <iostream>
00002 #include <algorithm>
00003 #include <string>
00004 #include <cstring>
00005
00006 using namespace std;
00007
00008 #include "simpleString.h"
00009
00010 #ifndef _MSC_FULL_VER // if not Visual Studio Compiler
00011     #warning "Klasa jest do zaimplementowania. Instrukcja w pliku naglowkowym"
00012 #else
00013     #pragma message ("Klasa jest do zaimplementowania. Instrukcja w pliku naglowkowym")
00014 #endif
```

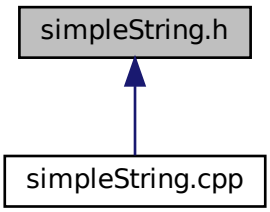
5.4 simpleString.h File Reference

Klasa do operowania na tekscie SimpleString: Proszę o napisanie klasy SimpleString opakowujacej dynamicznie allokowana tablice znakow. Klasa ta powinna zawierac (jako protected):

`#include <cstddef>`
Include dependency graph for simpleString.h:



This graph shows which files directly or indirectly include this file:



Classes

- class SimpleString

5.4.1 Detailed Description

Klasa do operowania na tekscie SimpleString: Proszę o napisanie klasy SimpleString opakowujacej dynamicznie allokowana tablice znakow. Klasa ta powinna zawierac (jako protected):

- skladowa statyczna: `std::size_t instances_` liczaca aktualnie zyjace instancje
- sugerowana skladowa: `char* data_` wskazująca na dynamicznie zaalokowany obszar z tekstem
- sugerowana skladowa: `std::size_t size_` zawierajaca informacje ile znakow trzyma aktualny tekst
- sugerowana skladowa: `std::size_t capacity_` zawierajaca informacje ile znakow pomiesci aktualny bufor bez realokazji

5.4.1.0.1 Poza skladowymi prosze o zaimplementowanie nastepujacych metod:

1. Metody stale `size()`, `capacity()`, `const char* data()` ktore zwroca powyzsze skladowe
   (a) size() ma zwracac rozmiar bez znaku konca tekstu
   (b) Prosze o napisanie metody stalej `const char* c_str()`, ktora zwroci zawartosc skladowej `data_`, ale zakonczona znakiem konca tekstu. W razie problemow mozna uzyc `mutable`.
2. Konstruktor bezargumentowy
   - wszystkie ustawienia na liście inicjalizacyjnej
   - standard dopuszcza `new char[0];`
3. Konstruktor przyjmujący tekst i dokonujący jego "głęboką" kopię
4. Konstruktor kopiujący, wykonujący "głęboką" kopię
5. Destruktor zwalniający pamięć
6. Metode statyczna `instances()`, ktora zwroci powyzsza skladowa statyczna
7. Proszę o napisanie metody `assign(const char* new_text)` ustawiajacej nowa zawartosc, dokonującej głębokiej kopii
   (a) **Prosze pamietac aby zwolnic stara pamiec!**
8. Prosze o napisanie metody `equal_to`, ktora przyjmie drugi SimpleString i sprawdzi czy zawieraja to samo.
   (a) Funkcja ta powinna przyjmowac jeszcze argument `bool case_sensitive=true` w oparciu o ktory porownanie bedzie ignorowac lub nie wielkosc znakow.
9. Prosze o napisanie metody `append`, ktora przyjmuje drugi SimpleString, a po jej zawolaniu jego zawartosc zostanie dodana do zawartosci `this`.

5.4.1.0.2 Uwaga1: Nie wolno uzywac: `std::string` ani `std::string_view` ani innych specjalizacji `std::basic_string<...>`!

5.4.1.0.3 Uwaga2: Prosze upewnic sie, ze nie ma wyciekow pamieci.

5.4.1.0.4 Prosze po zaimplementowaniu przypatrzyc sie dokumentacji `std::string` i porownac metody z `SimpleString`. Warto docenic, ze mamy `std::string` a wiec sami nie musimy sie bawic z pamiecia`

5.4.1.1 Do rozwazenia dla zaawansowanych:

1. Zaimplementowac copy-on-write - wtedy mozna te sugerowane skladowe zastapic.
2. W ktorych sytuacjach wygodnie byloby przeciazyc operatory?
3. Do czego jest `std::string_view`?
4. `std::string` ma definicje typu `value_type` - co nam to daje?
5. Prosze o zdefiniowanie metody `void load(std::istream& is)` (lub operator>>), ktora wczyta tekst dowolnej dlugosci z klawiatury
6. Jak zrobic metode `assign` aby uniknac niepotrzebnych alokacji i deallokacji pamieci?
7. Jak zrobic metody `assign` i `append` aby byly odporne na wyrzucenie wyjatku?
8. Jak wszystko zadzialaMimo iz mozna uzyc `std::unique_ptr`

5.4.1.2 Uwaga:

1. Wszystkie atrybuty powinny być prywatne, konstruktory i metody - publiczne
2. Metody większe niż 1-linijkowe powinny być zadeklarowane w klasie, zdefiniowane poza klasą w pliku zrodlowym
3. Obiekty typów klasowych powinny być w miarę możliwości przekazywane w argumentach funkcji przez referencję do stalej,
4. Proszę stosować słówko "const" w odpowiednich miejscach.
5. W pliku zrodlowym prosze nie wlaczac dodatkowych naglowkow typu: <iostream>, <algorithm> - takie rzeczy powinny byc w pliku zrodlowym
6. Prosze aby w pliku naglowkowym nie bylo `using namespace std;`, w zrodlowym moze.
7. Mozna korzystac z metod z <cstring>.
8. Jaka jest roznica miedzy `delete` a `delete []`?

Mozna tworzyc dowolna ilosc metod pomocniczych, jednakze aby byly one prywatne.

5.4.1.3 Punktacja:

Liczy sie przejście testów, aczkolwiek dobrze jakby tez nie bylo warningow i wyciekow pamieci
Definition in file simpleString.h.

5.5 simpleString.h

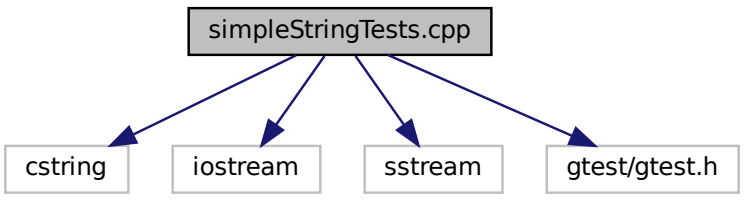
Go to the documentation of this file.

```
00001 #ifndef MATRIX_H
00002 #define MATRIX_H
00003
00076 #include <cstddef> // std::size_t
00077
00078
00079 class SimpleString
00080 {
00081 // TODO:
00082 };
00083
00084 #endif // MATRIX_H
```

5.6 simpleStringTests.cpp File Reference

```
#include <cstring>
#include <iostream>
#include <sstream>
#include <gtest/gtest.h>
```

Include dependency graph for simpleStringTests.cpp:



Classes

- struct SimpleStringTester

Functions

- void * operator new[] (size_t N)
- void operator delete[] (void *memory) noexcept
- TEST_F (SimpleStringTester, constructionWithDefaultConstructor_expectedAllElementsAreZero)
- TEST_F (SimpleStringTester, constructionFromText_expectedDataCopied)
- TEST_F (SimpleStringTester, constructionFromEmptyText_expectedEmptyTextCopied)
- TEST_F (SimpleStringTester, copyConstruction_expectedDataCopied)
- TEST_F (SimpleStringTester, destructor_expectedMemoryFreed)
- TEST_F (SimpleStringTester, instanceMethod_expectedCountedAllLivingInstances)
- TEST_F (SimpleStringTester, assign_expectedNewTextSet)
- TEST_F (SimpleStringTester, equalToWithoutIgnoringCases)
- TEST_F (SimpleStringTester, equalToWithDifferentCaseSettings)
- TEST_F (SimpleStringTester, appendOneTextAnotherOne_expectedBothTextMerged)

5.6.1 Function Documentation

5.6.1.1 operator delete[]()

```
void operator delete[] (
            void * memory )  [noexcept]
```

Definition at line 30 of file simpleStringTests.cpp.

5.6.1.2 operator new[]()

```
void * operator new[] (
            size_t N )
```

Definition at line 25 of file simpleStringTests.cpp.

5.6.1.3 TEST_F() [1/10]

```
TEST_F (
            SimpleStringTester ,
            appendOneTextAnotherOne_expectedBothTextMerged  )
```

inspiracja: https://www.sadistic.pl/cyps-albo-zyps-vt183512.htm
Definition at line 231 of file simpleStringTests.cpp.

5.6.1.4 TEST_F() [2/10]

```
TEST_F (
            SimpleStringTester ,
            assign_expectedNewTextSet  )
```

Definition at line 138 of file simpleStringTests.cpp.

5.6.1.5 TEST_F() [3/10]

```
TEST_F (
            SimpleStringTester ,
            constructionFromEmptyText_expectedEmptyTextCopied  )
```

Definition at line 65 of file simpleStringTests.cpp.

5.6.1.6 TEST_F() [4/10]

```
TEST_F (
            SimpleStringTester ,
            constructionFromText_expectedDataCopied  )
```

text from: https://demotywatory.pl/4925253/Jesli-cos-ci-w-zyciu-nie-idzie-i-wpadasz-pyskie
Definition at line 51 of file simpleStringTests.cpp.

5.6.1.7 TEST_F() [5/10]

```
TEST_F (
            SimpleStringTester ,
            constructionWithDefaultConstructor_expectedAllElementsAreZero  )
```

Definition at line 41 of file simpleStringTests.cpp.

5.6.1.8 TEST_F() [6/10]

```
TEST_F (
            SimpleStringTester ,
            copyConstruction_expectedDataCopied  )
```

Quote: Einstein:
Definition at line 78 of file simpleStringTests.cpp.

5.6.1.9 TEST_F() [7/10]

```
TEST_F (
            SimpleStringTester ,
            destructor_expectedMemoryFreed  )
```

to remove warning because of not used variable
Definition at line 98 of file simpleStringTests.cpp.

5.6.1.10 TEST_F() [8/10]

```
TEST_F (
            SimpleStringTester ,
            equalToWithDifferentCaseSettings  )
```

each block should be separate test!
Definition at line 210 of file simpleStringTests.cpp.

5.6.1.11 TEST_F() [9/10]

```
TEST_F (
            SimpleStringTester ,
            equalToWithoutIgnoringCases  )
```

each block should be separate test!
empty text
not empty text

different texts
different texts and different size of texts
Definition at line 167 of file simpleStringTests.cpp.

5.6.1.12 TEST_F() [10/10]

```
TEST_F (
            SimpleStringTester ,
            instanceMethod_expectedCountedAllLivingInstances  )
```

Definition at line 121 of file simpleStringTests.cpp.

5.7 simpleStringTests.cpp

Go to the documentation of this file.

```
00001 #include <cstring> // strlen()
00002
00003 #include <iostream>
00004 #include <sstream> // std::ostringstream
00005 #include <gtest/gtest.h>
00006
00007 #if __has_include("../simpleString.h")
00008     #include "../simpleString.h"
00009 #elif __has_include("simpleString.h")
00010     #include "simpleString.h"
00011 #else
00012     #error "File 'simpleString.h' not found!"
00013 #endif
00014
00015 using namespace std;
00016 using namespace ::testing;
00017
00018
00019 namespace
00020 {
00021     size_t allocations{};
00022     size_t deallocations{};
00023 } // namespace
00024
00025 void* operator new[](size_t N)
00026 {
00027     ++allocations;
00028     return malloc(N);
00029 }
00030 void operator delete[](void* memory) noexcept
00031 {
00032     ++deallocations;
00033     return free(memory);
00034 }
00035
00036
00037 struct SimpleStringTester : ::testing::Test
00038 {
00039 };
00040
00041 TEST_F(SimpleStringTester, constructionWithDefaultConstructor_expectedAllElementsAreZero)
00042 {
00043     SimpleString text;
00044
00045     ASSERT_EQ(0, text.size());
00046     ASSERT_EQ(0, text.capacity());
00047     ASSERT_STREQ("", text.data());
00048     ASSERT_STREQ("", text.c_str());
00049 }
00050
00051 TEST_F(SimpleStringTester, constructionFromText_expectedDataCopied)
00052 {
00054     constexpr const char* sourceText = "Jeśli coś ci w życiu nie idzie i wpadasz pyskiem w błoto, to
       musisz być jak dzik - pchać to błoto ryjem do przodu";
00055     const auto sourceTextSize = strlen(sourceText);
00056
00057     SimpleString text(sourceText);
00058
00059     ASSERT_EQ(sourceTextSize, text.size());
00060     ASSERT_LE(sourceTextSize, text.capacity());
00061     ASSERT_STREQ(sourceText, text.data());
00062     ASSERT_STREQ(sourceText, text.c_str());
00063 }
00064
```

```
00065 TEST_F(SimpleStringTester, constructionFromEmptyText_expectedEmptyTextCopied)
00066 {
00067     constexpr const char* sourceText = "";
00068     const auto sourceTextSize = strlen(sourceText);
00069
00070     SimpleString text(sourceText);
00071
00072     ASSERT_EQ(sourceTextSize, text.size());
00073     ASSERT_LE(sourceTextSize, text.capacity());
00074     ASSERT_STREQ(sourceText, text.data());
00075     ASSERT_STREQ(sourceText, text.c_str());
00076 }
00077
00078 TEST_F(SimpleStringTester, copyConstruction_expectedDataCopied)
00079 {
00081     constexpr const char* sourceText = "Wszystko powinno być tak proste, jak to tylko możliwe, ale nie
      prostsze";
00082     const auto sourceTextSize = strlen(sourceText);
00083
00084     SimpleString text1(sourceText);
00085     SimpleString text2(text1);
00086
00087     ASSERT_EQ(sourceTextSize, text1.size());
00088     ASSERT_LE(sourceTextSize, text1.capacity());
00089     ASSERT_STREQ(sourceText, text1.data());
00090     ASSERT_STREQ(sourceText, text1.c_str());
00091
00092     ASSERT_EQ(sourceTextSize, text2.size());
00093     ASSERT_LE(sourceTextSize, text2.capacity());
00094     ASSERT_STREQ(sourceText, text2.data());
00095     ASSERT_STREQ(sourceText, text2.c_str());
00096 }
00097
00098 TEST_F(SimpleStringTester, destructor_expectedMemoryFreed)
00099 {
00100     // source: https://www.cytaty.info/autor/terrypratchett-3.htm
00101     const char quote[] =    "Uniwersytety sa skarbnicami wiedzy: studenci przychodza ze szkol
      przekonani, \n"
00102                             "ze wiedza juz prawie wszystko; po latach odchodza pewni, ze nie wiedza
      praktycznie niczego. \n"
00103                             "Gdzie sie podziewa ta wiedza? Zostaje na uniwersytecie, gdzie jest
      starannie suszona \n"
00104                             "i skladana w magazynach. \n";
00105
00106     decltype(::allocations) allocationsBefore = ::allocations;
00107     decltype(::deallocations) deallocationsBefore = ::deallocations;
00108
00109     {
00110         SimpleString text(quote);
00111         static_cast<void>(text);
00112
00113         allocationsBefore = ::allocations;
00114         deallocationsBefore = ::deallocations;
00115     }
00116
00117     ASSERT_EQ(deallocationsBefore + 1, ::deallocations);
00118     ASSERT_LE(allocationsBefore, ::allocations);
00119 }
00120
00121 TEST_F(SimpleStringTester, instanceMethod_expectedCountedAllLivingInstances)
00122 {
00123     const auto instancesBefore = SimpleString::instances();
00124
00125     ASSERT_EQ(0, SimpleString::instances());
00126
00127     {
00128         SimpleString text1;
00129         ASSERT_EQ(1, text1.instances());
00130     }
00131     ASSERT_EQ(0, SimpleString::instances());
00132
00133     constexpr size_t instances2Create = 10;
00134     SimpleString textInstances[instances2Create];
00135     ASSERT_EQ(instances2Create, SimpleString::instances());
00136 }
00137
00138 TEST_F(SimpleStringTester, assign_expectedNewTextSet)
00139 {
00140     // source of quote: Sw. Antoni Pustelnik:
00141     const char quote1[] = "Przyjda takie czasy, ze ludzie beda szaleni "
00142                           "i gdy zobacza kogos przy zdrowych zmyslach, "
00143                           "powstana przeciw niemu mowiac: "
00144                           "Jestes szalony bo nie jestes do nas podobny.";
00145     const auto quote1Length = strlen(quote1);
00146
00147     // source: Sw. Jan Pawel II
00148     const char quote2[] = "Przyszlosc zaczyna sie dzisiaj, nie jutro";
```

```
00149     const auto quote2Length = strlen(quote2);
00150
00151     SimpleString text(quote1);
00152
00153     ASSERT_EQ(quote1Length, text.size());
00154     ASSERT_LE(quote1Length, text.capacity());
00155     ASSERT_STREQ(quote1, text.data());
00156     ASSERT_STREQ(quote1, text.c_str());
00157
00158
00159     text.assign(quote2);
00160
00161     ASSERT_EQ(quote2Length, text.size());
00162     ASSERT_LE(quote2Length, text.capacity());
00163     ASSERT_STREQ(quote2, text.data());
00164     ASSERT_STREQ(quote2, text.c_str());
00165 }
00166
00167 TEST_F(SimpleStringTester, equalToWithoutIgnoringCases)
00168 {
00170     {
00171         constexpr const char emptyText[] = "";
00172
00173         const SimpleString text1(emptyText);
00174         const SimpleString text2(emptyText);
00175
00176         ASSERT_STREQ(text1.c_str(), text2.c_str());
00177         ASSERT_TRUE(text1.equal_to(text2));
00178     }
00179
00180     {
00181         constexpr const char sourceText[] = ":)";
00182         const SimpleString text1(sourceText);
00183         const SimpleString text2(sourceText);
00184
00185         ASSERT_STREQ(text1.c_str(), text2.c_str());
00186         ASSERT_TRUE(text1.equal_to(text2));
00187     }
00188
00189     {
00190         constexpr const char sourceText1[] = ":)";
00191         constexpr const char sourceText2[] = ":(";
00192         const SimpleString text1(sourceText1);
00193         const SimpleString text2(sourceText2);
00194
00195         ASSERT_EQ(text1.size(), text2.size());
00196         ASSERT_FALSE(text1.equal_to(text2));
00197     }
00198
00199     {
00200         constexpr const char sourceText1[] = ":)";
00201         constexpr const char sourceText2[] = ":-)";
00202         const SimpleString text1(sourceText1);
00203         const SimpleString text2(sourceText2);
00204
00205         ASSERT_FALSE(text1.equal_to(text2));
00206     }
00207
00208 }
00209
00210 TEST_F(SimpleStringTester, equalToWithDifferentCaseSettings)
00211 {
00212     constexpr const char sourceText1[] = "To jest nasza ziemia!";
00213     constexpr const char sourceText2[] = "To jest nasza Ziemia!";
00214
00216     {
00217         const SimpleString text1(sourceText1);
00218         const SimpleString text2(sourceText2);
00219
00220         ASSERT_FALSE(text1.equal_to(text2, /*case_sensitive=*/true));
00221     }
00222
00223     {
00224         const SimpleString text1(sourceText1);
00225         const SimpleString text2(sourceText2);
00226
00227         ASSERT_TRUE(text1.equal_to(text2, /*case_sensitive=*/false));
00228     }
00229 }
00230
00231 TEST_F(SimpleStringTester, appendOneTextAnotherOne_expectedBothTextMerged)
00232 {
00234     constexpr const char sourceText1[] = "Cyps";
00235     constexpr const char sourceText2[] = " albo Zyps";
00236     string mergedText(sourceText1);
00237     mergedText.append(sourceText2);
00238
```

```
00239     const SimpleString text1(sourceText1);
00240     const SimpleString text2(sourceText2);
00241
00242     SimpleString textMerged(text1);
00243     textMerged.append(text2);
00244
00245     ASSERT_EQ(textMerged.size(), text1.size() + text2.size());
00246     ASSERT_STREQ(mergedText.c_str(), textMerged.data());
00247     ASSERT_STREQ(mergedText.c_str(), textMerged.c_str());
00248 }
```

Index